1

WRDC-TR-90-8007
Volume VIII
Part 29

INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume VIII - User Interface Subsystem
Part 29 - Text Editor Development Specification

F. Glandorf

Control Data Corporation
Integration Technology Services
2970 Presidential Drive
Fairborn, OH 45324-6209

September 1990

Final Report for Period 1 April 1987 - 31 December 1990

MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

[illegible] 25 July 91

DAVID L. JUDSON, Project Manager
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

DATE

FOR THE COMMANDER:

[illegible] 25 July 91

BRUCE A. RASMUSSEN, Chief
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

DATE

Unclassified

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution Is Unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) DS 620344600	5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-90-8007 Vol. VIII, Part 29
6a. NAME OF PERFORMING ORGANIZATION Control Data Corporation; Integration Technology Services 6b. OFFICE SYMBOL *(if applicable)*	7a. NAME OF MONITORING ORGANIZATION WRDC/MTI
6c. ADDRESS *(City, State, and ZIP Code)* 2970 Presidential Drive Fairborn, OH 45324-6209	7b. ADDRESS *(City, State, and ZIP Code)* WPAFB, OH 45433-6533
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Wright Research and Development Center, Air Force Systems Command, USAF 8b. OFFICE SYMBOL *(if applicable)* WRDC/MTI	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM. F33600-87-C-0464
8c. ADDRESS *(City, State, and ZIP Code)* Wright-Patterson AFB, Ohio 45433-6533	10. SOURCE OF FUNDING NOS.
11. TITLE *(Include Security Classification)* T See block 19	

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
78011F	595600	F95600	20950607

12. PERSONAL AUTHOR(S)
Structural Dynamics Research Corporation: Glandorf, F.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Yr.,Mo.,Day)*	15. PAGE COUNT
Final Report	4/1/87-12/30/90	1990 September 30	84

16. SUPPLEMENTARY NOTATION
WRDC/MTI Project Priority 6203

17. COSATI CODES

FIELD	GROUP	SUB GR.
1308	0905	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify block no.)*

19. ABSTRACT *(Continue on reverse if necessary and identify block number)*

This specification establishes the performance, development, test, and qualification requirements of the Text Editor computer program.

BLOCK 11:

INTEGRATED INFORMATION SUPPORT SYSTEM
Vol VIII -User Interface Subsystem

Part 29 - Text Editor Development Specification

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson	22b. TELEPHONE NO. *(Include Area Code)* (513) 255-7371 22c. OFFICE SYMBOL WRDC/MTI

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

FOREWORD

This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Branch Chief, Integration Technology Division, Manufacturing Technology Directorate, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W. A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. Jimmy P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

SUBCONTRACTOR	ROLE
Control Data Corporation	Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS.
D. Appleton Company	Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology.
ONTEK	Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use.
Simpact Corporation	Responsible for Communication development.
Structural Dynamics Research Corporation	Responsible for User Interfaces, Virtual Terminal Interface,and Network Transaction Manager design, development, implementation, and support.
Arizona State University	Responsible for test bed operations and support.

TABLE OF CONTENTS

TABLE OF CONTENTS(Continued)

TABLE OF CONTENTS(Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS

SECTION 1

SCOPE

1.1 Identification

This specification establishes the performance, development, test, and qualification requirements of a computer program identified as the Text Editor (TE). The Text Editor is one configuration item of the Integrated Information Support System (IISS) User Interface (UI).

Please refer to the Software Availability Bulletin, Volume III, Part 16, CI# SAB620326000, for current IISS software and documentation availability.

1.2 Functional Summary

The Text Editor provides the user with the ability to edit a file in a manner consistent with the User Interface. It also provides some extensions to the User Interface Form Processor for text editing within form data areas. Editing functions such as inserting, deleting, moving and replacing text are available.

SECTION 2

DOCUMENTS

2.1 Reference Documents

[1] Systran, ICAM Documentation Standards, IDS150120000C, 15 September 1983.

[2] Structural Dynamics Research Corporation, Virtual Terminal Development Specification, DS 620344300, 31 May 1988.

[3] Structural Dynamics Research Corporation, Form Processor Development Specification, DS 620344200, 31 May 1988.

[4] Structural Dynamics Research Corporation, User Interface Development Specification, DS 620344100, 31 May 1988.

[5] Structural Dynamics Research Corporation, Application Interface Development Specification, DS 620344700, 31 May 1988.

[6] Structural Dynamics Research Corporation, Rapid Application Generator Development Specification, DS 620344502, 31 May 1988.

[7] Structural Dynamics Research Corporation, Forms Driven Form Editor Development Specification, DS 620344402, 31 May 1988.

[8] Structural Dynamics Research Corporation, Forms Language Compiler Development Specification, DS 620344401, 31 May 1988.

[9] General Electric CO., System Design Specification, 7 February 1983.

2.2 Terms and Abbreviations

Buffer Name: the default file in which the buffer will be saved if no file is given on a save command.

<u>Current Cursor Position</u>: the position of the cursor before an edit command or function is issued in the text editor.

<u>Cursor Position</u>: the position of the cursor after any command is issued.

<u>Cut and Paste Buffer</u>: where deleted lines go and the paste and fill edit commands get their data.

<u>Display Start Line</u>: the first line in the buffer to be displayed.

<u>Display Size</u>: the number of lines used in the edit area.

<u>Field</u>: two dimensional space on a terminal screen.

<u>Field Pointer</u>: indicates the ITEM which contains the current cursor position.

<u>Form Processor</u>: (FP), subset of the IISS User Interface that consists of a set of callable execution time routines available to an application program for form processing.

<u>Form Processor Text Editor</u>: (FPTE), subset of the Form Processor that consists of software modules that provide text editing capabilities to all users of applications that use the Form Processor.

<u>IISS Function Screen</u>: the first screen that is displayed after logon. It allows the user to specify the function he wants to access and the device type and device name on which he is working.

<u>Integrated Information Support System</u>: (IISS), a computing environment used to investigate, demonstrate, test the concepts and produce application for information management and information integration in the context of Aerospace Manufacturing. The IISS addresses the problems of integration of data resident on heterogeneous data bases supported by heterogeneous computers interconnected via a Local Area Network.

<u>Item</u>: non-decomposable area of a form in which hard-coded descriptive text may be placed and the only defined areas where user data may be input/output.

Message: descriptive text which may be returned in the standard message line on the terminal screen. They are used to warn of errors or provide other user information.

Message Line: a line on the terminal screen that is used to display messages.

Operating System: (OS), software supplied with a computer which allows it to supervise its own operations and manage access to hardware facilities such as memory and peripherals.

Paging and Scrolling: a method which allows a form to contain more data than can be displayed with provisions for viewing any portion of the data buffer.

Previous Cursor Position: the position of the cursor when the previous edit command was issued.

Previous Edit Command: the function key pressed before the current one.

Select Line: one terminus of the select range.

Select Mode: when on, certain commands will be executed over the lines in the selected range. The commands are <DELETE LINE> and replace.

Text Editor: (TE), subset of the IISS User Interface that consists of a file editor that is based on the text editing functions built into the Form Processor.

Top of file: the first line of the buffer.

User Interface: (UI), IISS subsystem that controls the user's terminal and interfaces with the rest of the system. The UI consists of two major subsystems: the User Interface Development System (UIDS) and the User Interface Management System (UIMS).

User Interface Development System: (UIDS), collection of IISS User Interface subsystems that are used by applications programmers as they develop IISS applications. The UIDS includes the Form Editor and the Application Generator.

User Interface Services: (UIS), subset of the IISS User Interface that consists of a package of routines that aid users in controlling their environment. It includes message management, change password, and application definition services.

Virtual Terminal: (VT), subset of the IISS User Interface that performs the interfacing between different terminals and the UI. This is done by defining a specific set of terminal features and protocols which must be supported by the UI software which constitutes the virtual terminal definition. Specific terminals are then mapped against the virtual terminal software by specific software modules written for each type of real terminal supported.

SECTION 3

REQUIREMENTS

3.1 Computer Program Definition

The Text Editor is an IISS application and as such can be invoked from the IISS function menu. It provides IISS users with file editing capabilities.

The Text Editor also works in conjunction with the User Interface Form Processor (FP) to provide the ability to edit ITEMs displayed on a form. In this mode it is referred to as the Form Processor Text Editor (FPTE) and can be invoked by the terminal user of any application that uses the Form Processor. Editing functions such as inserting, deleting, moving, and replacing text are available. Some commands are not available in the FPTE and others have a slightly different meaning. This is because when editing a file, the file is placed in a buffer of which a portion is displayed on the screen where as with an ITEM all of its data are displayed. The differences in commands are due to file manipulation and requirements to display various parts of the file buffer.

The Text Editor has two basic modes, edit and command. Usually you are in the edit mode entering text and using key commands, called functions. The command mode is entered by pressing the <COMMAND> key in the edit mode. The command mode is exited by completing a command or by pressing the <QUIT> key.

Help is available for both modes.

The conditions under which an error occurs and its handling are listed under each function. In general if an error condition occurs the function is aborted and a message is displayed warning of the failure.

3.1.1 Interface Requirements

3.1.1.1 Interface Block Diagram

Figure 3-1 describes the structure of the TE interfaces.

```
+---------+
|   TE    +------------------------+
+---------+                        |
|   AI    |                        | OS file access method
+---+-----+                        |
    |                          +---+---+
    |                          | file  |
    |                          +-------+
+---+-----+
|  NTM    |
+---+-----+
    |
+---+-----+
|  UIM    |
+---------+
|   FP    |
+---------+
```

Figure 3-1 Text Editor Interfaces

3.1.1.2 Detailed Interface Definition

The TE is really two entities. As an application function, it is capable of editing files. In this role it is a form processor application. The FPTE is a mode of the Form Processor. In this role it is available for editing ITEMs of any application. They have a common set of functions and commands.

The Text Editor, when run as a Form Processor application, is invoked by filling in the FUNCTION item on the IISS function form with SDTEZZZZZZ. As a Form Processor application it has been linked using the Application Interface routines. The Text Editor, when run as a mode of the Form Processor, is invoked by pressing the <MODE> key until "text editor" appears in the bottom right hand corner of the screen.

The Text Editor may also be invoked from another application. Two parameters must be passed to EDITCI routine: first the name of the file to be edited (up to 30 characters) and second, a flag variable which is set to TRUE if the file was written, otherwise FALSE.

3.2 Detailed Functional Requirements

3.2.1 File Editing

3.2.1.1 Edit Mode

In addition to being able to enter text in the edit mode the following commands are available through the function keys. A phrase enclosed in angle brackets "<>", denotes a key. Some commands are implemented through the Virtual Terminal (VT). They are listed below:

<UP ARROW> -	Moves the cursor up one line.
<DOWN ARROW> -	Moves the cursor down one line. Note that these keys do not scroll the screen. The scroll keys do that.
<LEFT ARROW> -	Moves the cursor left one character.
<RIGHT ARROW> -	Moves the cursor right one character. Note that the cursor will not "wrap around".
<TAB> -	Moves the cursor to the start of the next line. Except when on the next to last line when the cursor will move to the message number field (MSG:).
<BACK TAB> -	Moves the cursor to the start of the previous line. Except when on the first line when the cursor will move to the message number field (MSG:).
<DELETE CHARACTER>-	Shift left one character the character from just right of the cursor to the end of the line. The right most character in the line is replaced with a blank.

<INSERT CHARACTER>-	Shift right one character the character from the cursor position to the end of the field line. The right most character in the line are lost and the vacated character position is filled with a blank.

The following group of functions are implemented through the TE itself and accessible by function keys on the terminal. Appendix C contains keypad layouts for a VT100 terminal.

<COMMAND MODE>	
<DELETE LINE>	
<INSERT LINE>	
<NEXT LINE>	- In scroll/page mode use vertical scroll up (see Terminal Operator's Guide on paging and scrolling).
<PREVIOUS LINE>	- In scroll/page mode use vertical scroll down (see Terminal Operator's Guide on paging and scrolling).
<NEXT PAGE>	- In scroll/page mode use vertical page up (see Terminal Operator's Guide on paging and scrolling).
<PREVIOUS PAGE>	- In scroll/page mode use vertical page down (see Terminal Operator's Guide on paging and scrolling).
<FIRST PAGE>	
<LAST PAGE>	
<MIDLINE BREAK>	
<SELECT>	
<PASTE>	
<FILL>	
<QUIT>	
<SEARCH NEXT>	

Figure 3-2 Shows the editing form. For a VT100 the top 22 lines are used for the editing area. The line marked with the ">>" is the command line.

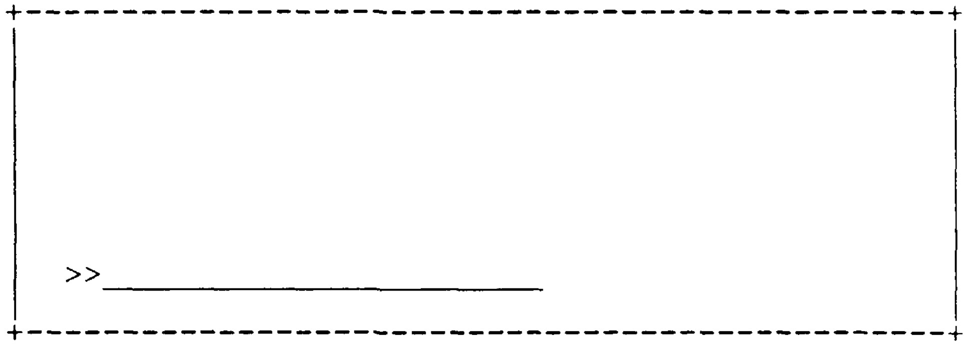


Figure 3-2 Edit Mode Form

3.2.1.1.1 Edit to Command Mode

<COMMAND>	- In the edit mode this is used to switch to the command mode.

There are no error conditions with this function.

3.2.1.1.2 Scroll Up

<SCROLL UP(FP FUNCTION)> - Scroll the displayed portion of the buffer up by one line. Push the top displayed line into the nondisplayed portion of the buffer and display the line in the buffer which follows the last line displayed.

If the display start line is the bottom of the file the function is aborted and a message warning of the condition is issued.

In Appendix A see figure A-5 (before) and A-6 (after) a Next Line (Scroll Up).

3.2.1.1.2.1 Input

Top of File.
Bottom of File.
Display Start Line.
Current Cursor Position.

3.2.1.1.2.2 <u>Output</u>

Display Start Line	-	This is moved to the next line.
Cursor Position	-	This is moved to the next line.

3.2.1.1.3 <u>Scroll Down</u>

<SCROLL DOWN(FP FUNCTION)> - This is similar to <SCROLL UP> except that the display is scrolled down.

If the display start line is just after the top of file mark, then the function is aborted and a warning message is issued.

In Appendix A see figure A-7 (before) and A-8 (after) a Previous Line (Scroll Down).

3.2.1.1.3.1 <u>Input</u>

Top of File.
Bottom of File.
Display Start Line.
Current Cursor Position.

3.2.1.1.3.2 <u>Output</u>

Display Start Line	-	This is moved to the previous line in the TE buffer.
Cursor Position	-	This is moved to the previous line in the TE buffer.

3.2.1.1.4 <u>Page Up</u>

<PAGE UP (FP FUNCTION)> - The display is scrolled up by the display size.

If the display start line is the bottom of file mark then the function is aborted and a warning message is issued.

In Appendix A see figure A-9 (before) and A-10 (after) a Next Page (Page Up).

3.2.1.1.4.1 <u>Input</u>

Top of File.
Bottom of File.
Display Start Line.
Current Cursor Position.

3.2.1.1.4.2 <u>Output</u>

Display Start Line	-	This is moved forward in the TE buffer by display size lines.
Cursor Position	-	This is moved forward in the TE buffer by display size lines.

3.2.1.1.5 <u>Page Down</u>

<PAGE DOWN (FP FUNCTION)> - The display is scrolled down by the display size.

If the display start line is just after the top of file mark then the function is aborted and a warning message is issued.

In Appendix A see figure A-11 (before) and A-12 (after) a Previous Page (Page Down).

3.2.1.1.5.1 <u>Input</u>

Top of File.
Bottom of File.
Display Start Line.
Current Cursor Position.

3.2.1.1.5.2 <u>Output</u>

Display Start Line	-	This is moved backward in the TE buffer by display size lines.
Cursor Position	-	This is moved backward in the TE buffer by display size lines.

3.2.1.1.6 <u>First Page</u>

<FIRST PAGE>	-	Displays the first display size lines of the buffer.

There are no error conditions with this function.

In Appendix A see figure A-13 (before) and A-14 (after) a First Page.

3.2.1.1.6.1 Input

Top of File.
Bottom of File.
Display Start Line.
Current Cursor Position.

3.2.1.1.6.2 Output

Display Start Line	-	This is moved to Top of File.
Cursor Position	-	This is moved to Top of File.

3.2.1.1.7 Last Page

<LAST PAGE>	-	Displays the last display size lines of the buffer.

There are no error conditions with this function.

In Appendix A see figure A-15 (before) and A-16 (after) a Last Page.

3.2.1.1.7.1 Input

Top of File.
Bottom of File.
Display Start Line.
Current Cursor Position.

3.2.1.1.7.2 Output

Display Start Line	-	If there are more than display size lines in the TE buffer this is moved to a line display size backward from the bottom of file. If there are fewer than display size lines in the TE buffer this is moved to Top of file.
Cursor Position	-	This is moved to bottom of file.

3.2.1.1.8 Insert Line

<INSERT LINE>	-	Move down one line the line containing the cursor and all following lines. The last line is pushed into the nondisplayed portion of the buffer and the line containing the cursor is filled with blanks.

The TE does not permit insert lines to be done on lines following the bottom of file mark. An attempt to do so aborts the function and a warning message is issued. If the TE is unable to allocate sufficient memory for the new line a message is displayed warning of the condition and the operation is aborted. Since many functions use insert line this last condition may occur in several places: midline break, select, paste, fill, and load a file. It is expected that this will be a rare condition.

In Appendix A see figure A-3 (before) and A-4 (after) an <INSERT LINE>.

3.2.1.1.8.1 Input

Top of File.
Bottom of File.
Display Start Line.
Current Cursor Position.

3.2.1.1.8.2 Output

Display Start Line	-	If the cursor was on the display start line prior to the function then the new display start line is the line that was just inserted.
Cursor Position	-	The cursor is moved the the line that was just inserted (i.e. the same physical line on the screen).

3.2.1.1.9 Midline Break

<MIDLINE BREAK>	-	All characters from the cursor position to the end of the line are removed and inserted at the beginning of a newline immediately following the cursor's line.

A midline break is not permitted on the bottom of file mark or any following line. If attempted the function is aborted and a warning is issued. If the insert line fails the midline break will be aborted as well.

In Appendix A see figure A-17 (before) and A-18 (after) a Midline Break.

3.2.1.1.9.1 Input

Top of File.
Bottom of File.
Display Start Line.
Current cursor Position.

3.2.1.1.9.2 Output

Cursor position	-	The cursor is moved to the start of the newline.

3.2.1.1.10 Cut and Paste Scenario

Move the cursor to the first line to move and press the <DELETE LINE> key for the n consecutive lines to be moved. Alternately, to move large blocks of text, move the cursor to the first line to move and press <SELECT> then move the cursor to the last line to move and press <DELETE LINE>. Next move the cursor to the line where the lines are to be pasted then press the <PASTE> key or <FILL> key. The <PASTE> key will restore the deleted lines in exactly the same format they were when deleted. The <FILL> key will restore the deleted lines and fill each newline to the extent of the current fill margins with as many words as possible from the paste buffer. A word is a sequence of nonblank printable characters. Note that the lines remain in the cut and paste buffer until another <DELETE LINE> is issued so the lines may be pasted in as many places as required. The <SELECT> mode may be canceled by pressing <SELECT> again without pressing <DELETE LINE>.

In Appendix A see figures A-19,20,21, and 22 for how the lines are put into the cut and paste buffer, figures A-23 and 24 for a paste, and figures a-25 and 26 for a paste with fill (margins at 40 and 70).

3.2.1.1.10.1 Select for Cut and Paste

<SELECT>	- A <MIDLINE BREAK> is performed and the line after the cursor is marked as one terminus of the select range for deletion. This position is marked on the screen with a special line. If the select mode is currently on, it is canceled.

Select is not permitted on the bottom of file mark or any following lines. An attempt to do so will abort the function and a warning message will be issued. Select uses insert line to display the select marker. If the insert line fails the select is aborted.

In Appendix A see figure A-19 (before) and A-20 (after) a Select.

3.2.1.1.10.1.1 Input

Current Cursor Position.
Select Mode.

3.2.1.1.10.1.2 Output

Cursor Position	- If the select mode is off, the cursor is moved to the start of line containing the select line marker. If the select mode is on the cursor position is unchanged.
Select Mode	- Toggled from its input status.
Select Line Terminus	- Indicates a position in the TE buffer as a terminus for deletion.

3.2.1.1.10.2 Delete a Line of Text

<DELETE LINE>	- The characters from the cursor position to the end of the line are deleted. The cursor is moved to the start of the next line. If the cursor was at the beginning of the line all lines which follow are scrolled up one. One of the following things happen to the line:

1. If the select mode is on, all the lines in the selected range replace the contents of the cut and paste buffer.

2. If the select mode is off and the previous function was a delete and the user has not moved the cursor, the line is concatenated to the cut and paste buffer.

3. If the select mode is off and the previous function is not the same as the current (<DELETE LINE>) or the user has moved the cursor, the line replaces the contents of the cut and paste buffer.

The TE does not permit the line containing the bottom of file marker or any following lines to be deleted. An attempt to do so is aborted and a warning message is issued. The cursor may be in column one of the line containing the bottom of file marker if the select mode is on.

In Appendix A see figure A-1 (before) and A-2 (after) a <DELETE LINE>.

3.2.1.1.10.2.1 <u>Input</u>

Top Of File	-	First line of text in TE buffer.
Bottom Of File	-	Last line of text in the TE buffer.
Display start line	-	The first line of the TE buffer to be displayed.
Previous Edit Command	-	This is required for the cut and paste operation (see below).
Previous Cursor Position	-	This is required for the cut and paste operation.
Select Mode	-	If this mode is on then all lines in the select range are deleted.
Select Line	-	If the select mode is on then all lines between the selected line and the current cursor line inclusive are deleted.
Current Cursor position	-	This is the line the cursor is currently on.

Cut and Paste Buffer.

3.2.1.1.10.2.2 Output

Display Start Line-	If the cursor was on the Display start line then the display start line is move to the next line.
Cursor Position -	The cursor will be positioned at the beginning of the line which follows the deleted line(s). This will enable a <PASTE> or <FILL> command to be issued immediately afterward without the user having to move the cursor.
Select Mode -	Set to False.

Cut and Paste Buffer.

3.2.1.1.10.3 Paste for Cut and Paste

<PASTE> -	This will insert the contents of the cut and paste buffer (exactly as they appear in that buffer) into the TE buffer just before the line containing the current cursor position.

A paste is not permitted on any line following the bottom of file mark. An attempt to do so will abort the function and a warning message will be issued. Since paste uses insert line if it fails then paste will be aborted.

In Appendix A see figure A-23 (before) and A-24 (after) a Paste.

3.2.1.1.10.3.1 Input

Top of File.
Bottom of File.
Current Cursor Position.
Cut and Paste Buffer.
Display Start Line.

3.2.1.1.10.3.2 Output

Display Start Line -	This is moved so that only lines above the cursor position are changed.

3.2.1.1.10.4 Fill for Cut and Paste

<FILL>	- The contents of the cut and paste buffer are inserted into the TE buffer just before the line containing the current cursor position. Unlike <PASTE>, <FILL> will place as many whole words on each line within the current fill margins. A word is defined as a sequence of nonblank printable characters.

If a word is to large to fit within these margins it is wrapped and a warning message is issued. A fill is not permitted on any line after the bottom of file mark. An attempt to do so will be aborted and a warning message will be issued. Since fill uses insert line if it fails then fill will be aborted.

In Appendix A see figure A-25 (before) and A-26 (after) a Fill.

3.2.1.1.10.4.1 Input

Top of File.
Bottom of File.
Current Cursor Position.
Cut and Paste Buffer.
Display Start Line.
Fill Margins - Two values which satisfy the following constraint:
0 <= left margin < right margin <= display width.

3.2.1.1.10.4.2 Output

Display Start Line	- This is moved so that only lines above the cursor position are changed.
Cursor Position	- Moved to the line following the pasted lines.

3.2.1.1.11 Quit the TE

<QUIT>	- Terminates the edit or command mode. If in the edit mode and the buffer has been modified since the initial load or last save, you must press the <QUIT> key

twice. Pressing another function key the second time returns to the edit mode. If in the command mode it returns to the edit mode without executing a command.

There are no error conditions with this function.

3.2.1.1.11.1 Input

TE buffer change flag	- True if the buffer has been modified since it was loaded or the last save.

3.2.1.1.12 Search Next

<SEARCH NEXT>	- The most recently executed search command will be repeated. The error conditions are the same as for the <SEARCH> function.

3.2.1.1.12.1 Input

Current cursor position.
Bottom of File.
String.
Direction.

3.2.1.1.12.2 Output

Cursor Position.
Display Start Line.

3.2.1.2 Functionality of Command Mode Commands

These are accessed by pressing the <COMMAND> key while in the edit mode. The cursor then moves to the command line. At this point you can enter the command with its operands and press <ENTER> or leave the field blank and press <HELP> OR <ENTER>. This second option displays a menu. You then move the cursor to the field with the desired command, and either type in the operands and press <ENTER> or <HELP> and a form with fields for the operands is displayed, press <ENTER> to enter this form. The command mode is exited by completing a command or pressing the <QUIT> key. If the command fails, you remain in the current command mode (form, menu or command line). In the following, references to the cursor position are to the cursor's position in the buffer before the <COMMAND> key was pressed. Each command or operand is separated by a blank. Operands which contain a blank must be enclosed in double quotes ("). On successful completion, the message "operation complete" is issued. A list of commands follows:

SEARCH
REPLACE
LOAD
SAVE
CLEAR
MARGINS
REPEAT
BUFFER

Figure 3-3 shows the Command Menu you get on pressing <ENTER> to the command line. A command's operands may be entered or a blank line. Entering a blank line causes the individual command's form to be displayed. This form is displayed at the bottom of this form.

```
+---------------------------------------+
|                                       |
|       Command Menu                    |
|        Search: ___________________    |
|       Replace: ___________________    |
|          Load: ___________________    |
|          Save: ___________________    |
|         Clear: ___________________    |
|       Margins: ___________________    |
|        Repeat: ___________________    |
|        Buffer: ___________________    |
|                                       |
|                                       |
|                                       |
|                                       |
+---------------------------------------+
```

Figure 3-3 Command Menu

3.2.1.2.1 Search for a Given String

SEARCH string
direction - Finds the first occurrence of the string

after the cursor position and positions the cursor to the first character of the string in the buffer. If the command is appended with a minus (e.g. search string -), the search will be done "backwards". If the string is not found a message is displayed and you remain in the current command mode. If no operand is supplied with search, the last string and direction searched for is used.

A search is not permitted on the bottom of file mark or any following line. An attempt to do so aborts the command and issues a warning message.

Figure 3-4 shows the search for a string form.

```
+----------------------------------------+
|                                        |
|       Command Menu                     |
|       Search: ___________________      |
|      Replace: ___________________      |
|         Load: ___________________      |
|         Save: ___________________      |
|        Clear: ___________________      |
|      Margins: ___________________      |
|       Repeat: ___________________      |
|       Buffer: ___________________      |
|                                        |
|  Search for a String                   |
|  Enter String: __________________      |
|  Search Direction: ______________      |
|                                        |
+----------------------------------------+
```

Figure 3-4 Search for a String

3.2.1.2.1.1 Input

Current Cursor Position.		
Bottom of File.		
String	-	This is the string to search the TE buffer for. If no string is supplied

the last searched for string will be used.

Direction	-	The direction of the search.

3.2.1.2.1.2 Output

Cursor Position.
Display start line.

3.2.1.2.2 Replace One Text String with Another

REPLACE "fromstr" "tostr" option direction - Finds the first occurrence of the fromstr starting with the cursor position and replaces it with the tostr. If a star is appended to the command (e.g. fromstr tostr *), the replacements are made on all occurrences from the cursor to the end of the buffer. If a dot "." is appended to the command, the replacements are made on all occurrences from the cursor to the end of the line. A star and a dot may not be used in the same command. If a minus "-" is appended to one of the above commands the replacements will be performed "backwards". For regular commands the first occurrence to the left of the cursor is replaced. For commands with a dot all occurrences from the beginning of the line to the cursor are replaced. For commands with a star all occurrences from the top of file to the cursor are replaced. If the select range is active, then replacements are performed on all lines in this range and the select mode is turned off. If the fromstr is not found, a message is displayed and you remain in the current command mode.

The cursor is positioned just after the replaced text for regular commands. For commands appended with a dot, the cursor is left at column one of the next line. For commands appended with a star, the cursor is positioned at the bottom of the file mark or at the end of the select range if that form of the command is used. For commands appended with a minus and for regular commands, the cursor is positioned at the left most column of the to string. For commands appended with a dot, the cursor is positioned at column one of the line. For commands appended with a star, the cursor is positioned at column one of the top of the file line. A line that is longer after the replacement is wrapped, and a shorter line is padded on the right with blanks.

A replace is not permitted on the bottom of file line or any following line. An attempt to do so will abort the command and a warning message is issued.

Figure 3-5 shows the replace string form.

```
+----------------------------------------+
|                                        |
|       Command Menu                     |
|       Search: ___________________      |
|      Replace: ___________________      |
|         Load: ___________________      |
|         Save: ___________________      |
|        Clear: ___________________      |
|      Margins: ___________________      |
|       Repeat: ___________________      |
|       Buffer: ___________________      |
|                                        |
|  Replace One String With Another       |
|  Enter Old String: ______________      |
|  Enter New String: ______________      |
|            Option: ______________      |
|  Direction: ____                       |
+----------------------------------------+
```

Figure 3-5 Replace String

3.2.1.2.2.1 Input

Current Cursor Position.
Bottom of File.

From string	-	This is the text string to be replaced with the To string.
To string	-	This is the text to replace each occurrence of the From string.
Option	-	blank: first occurrence, dot: to end of line, star: to end of file.
Direction	-	blank: forward, minus backward.

Select Mode.

3.2.1.2.2.2 Output

Cursor Position.
Display line start.

3.2.1.2.3 Load a File for Editing

LOAD file	-	The file is inserted into the buffer

just before the line containing the cursor.

A load is not permitted if the current cursor position is below the bottom of file mark. An attempt to do so aborts the command and a warning message is issued. The following I/O operations may cause the command to be aborted and a warning message issued: Open file, Close file, and Read a record. Since LOAD uses insert line, its failure causes the LOAD to be aborted.

In appendix A see figure A-27 (before) and A-28 (after) a Load.

Figure 3-6 shows the command form with the load form.

```
+----------------------------------------+
|                                        |
|       Command Menu                     |
|       Search: ___________________      |
|      Replace: ___________________      |
|         Load: ___________________      |
|         Save: ___________________      |
|        Clear: ___________________      |
|      Margins: ___________________      |
|       Repeat: ___________________      |
|       Buffer: ___________________      |
|                                        |
|      Load a File                       |
|                                        |
|  Enter File Name: _______________      |
|                                        |
+----------------------------------------+
```

Figure 3-6 Load a File

3.2.1.2.3.1 Input

File name	-	This is the file to be loaded for editing.

Current Cursor Position.

3.2.1.2.3.2 Output

Display Start Line	-	This is moved so that only lines above the cursor position are changed.
Cursor Position	-	Moved to the line following the loaded lines.

3.2.1.2.4 Save the TE Buffer

SAVE file	-	The contents of the buffer are written to the file. If no file name is supplied, the buffer is written to the current buffer name.

The following I/O operations may cause the SAVE command to be aborted and a warning message issued: Open file, Close file, and Write a record. The state of the file on such a failure is system dependent. If no file name is supplied and there is no buffer name, a message is displayed and you remain in the current command mode.

Figure 3-7 shows the save the buffer to a file form.

```
+----------------------------------------+
|                                        |
|       Command Menu                     |
|        Search: ___________________     |
|       Replace: ___________________     |
|          Load: ___________________     |
|          Save: ___________________     |
|         Clear: ___________________     |
|       Margins: ___________________     |
|        Repeat: ___________________     |
|  Save a File                           |
|                                        |
| Enter File Name: ________________      |
|                                        |
+----------------------------------------+
```

Figure 3-7 Save Buffer

3.2.1.2.4.1 Input

File name	- This is the name of the file to which the contents of the TE buffer is written. If no file name is given in the command, the buffer's name is used.

Buffer name.
Top of File.
Bottom of File.

3.2.1.2.5 Clear the TE Buffer

CLEAR	- The buffer is cleared of text by deleting all lines between top of file and bottom of file. The lines replace the contents of the cut and paste buffer. Clear has no operands.

There are no error conditions with this command.

Figure 3-8 shows the clear buffer form.

```
+----------------------------------------+
|                                        |
|        Command Menu                    |
|        Search: ___________________     |
|       Replace: ___________________     |
|          Load: ___________________     |
|          Save: ___________________     |
|         Clear: ___________________     |
|       Margins: ___________________     |
|        Repeat: ___________________     |
|        Buffer: ___________________     |
|                                        |
|    Clear Buffer                        |
|                                        |
|    Delete all Lines? ____              |
|                                        |
+----------------------------------------+
```

Figure 3-8 Clear the Buffer

3.2.1.2.5.1 <u>Input</u>

Top of File.
Bottom of File.

3.2.1.2.5.2 <u>Output</u>

Display start line.
Cursor position.

3.2.1.2.6 <u>Set Limits of the Fill Margins</u>

FILL MARGINS "left margin" "right margin" - Sets the left and right margins for the <FILL> key. The margin values must satisfy the following constraint:

0 <= "left margin" < "right margin" <= display size + 1.

If the above constraint is not satisfied, the command is aborted and a warning message is issued.

Figure 3-9 shows the Set Fill Margins form.

```
+----------------------------------------+
|                                        |
|       Command Menu                     |
|        Search: ___________________     |
|       Replace: ___________________     |
|          Load: ___________________     |
|          Save: ___________________     |
|         Clear: ___________________     |
|       Margins: ___________________     |
|        Repeat: ___________________     |
|        Buffer: ___________________     |
|                                        |
|  Set Fill Margins                      |
|  Left Margin: ___                      |
|  Right Margin: ___                     |
|                                        |
+----------------------------------------+
```

Figure 3-9 Set Fill Margins

3.2.1.2.6.1 Input

Left margin	-	The left margin extends from zero to this number.
Right margin	-	The right margin extends from this value to the display size plus one.

3.2.1.2.7 Repetition of a Command

REPEAT number	-	This allows some function keys and all commands to be repeated number times. Upon entering this command, the message "in repeat" is displayed. Enter the command or press the function key which is to be repeated. The function keys which are supported include: <INSERT LINE>, <DELETE LINE>, <PASTE>, and <FILL>. The <FIRST PAGE>, <LAST PAGE> and <QUIT> keys are executed only once.

The number of repetitions must be greater than zero. If not, the function is aborted and a warning message is issued.

Figure 3-10 shows the Repeat Command/Function form.

```
+----------------------------------------+
|                                        |
|         Command Menu                   |
|          Search: ___________________   |
|         Replace: ___________________   |
|            Load: ___________________   |
|            Save: ___________________   |
|           Clear: ___________________   |
|         Margins: ___________________   |
|          Repeat: ___________________   |
|          Buffer: ___________________   |
|                                        |
|    Repeat Command/Function             |
|    Enter Repeat: ____                  |
|                                        |
|                                        |
+----------------------------------------+
```

Figure 3-10 Repeat Command/Function

3.2.1.2.7.1 <u>Input</u>

Number	- This is the number of times the command is repeated.
Command	- This is the command to execute number times.

Other inputs to this command are determined by the repeated command.

3.2.1.2.7.2 <u>Output</u>

Outputs of this command are determined by the repeated command.

3.2.1.2.8 <u>Name the Buffer</u>

Buffer "filename"-	This becomes the default name for a save command.

There are no error conditions associated with this command.

Figure 3-11 shows the name the buffer form.

```
+----------------------------------------+
|                                        |
|         Command Menu                   |
|          Search: ___________________   |
|         Replace: ___________________   |
|            Load: ___________________   |
|            Save: ___________________   |
|           Clear: ___________________   |
|         Margins: ___________________   |
|          Repeat: ___________________   |
|          Buffer: ___________________   |
|                                        |
|    Buffer Name                         |
|                                        |
|    Enter Buffer Name: _____________    |
|                                        |
+----------------------------------------+
```

Figure 3-11 Name Buffer

3.2.1.2.8.1 Output

Buffer name.

3.2.2 ITEM Editing in the Form Processor Mode

The purpose of ITEM Editing is to facilitate moving, copying, deleting and substituting text among ITEMS. The FP/TE understands the structure of forms sufficiently to allow this to be done in a friendly manner. ITEM editing is a submode of the FP. You enter the FP/TE by pressing the FP <MODE> key until the "text edit" choice appears in the mode field. You return to the application by pressing the <MODE> key until the "application" choice appears in the mode field. All functions and commands work with ITEMs only so the cursor must be within an ITEM when they are issued. The VT functions which are exceptions are noted.

Error handling is listed under each function. In general an error causes the function to be aborted and a warning message to be issued. If the cursor is not in an ITEM when a function or command is issued, the command is aborted and a warning message issued.

3.2.2.1 Commands Implemented Through Keys

The following is a list of functions which are implemented through the VT. The VT functions which are usable outside of an ITEM are marked with a star "*".

<UP ARROW>*	-	Mcve cursor up one line.
<DOWN ARROW>*	-	Move cursor down one line.
<LEFT ARROW>*	-	Move cursor left one character.
<RIGHT ARROW>*	-	Move cursor right one character.
<TAB>*	-	Moves cursor to the next input ITEM.
<BACK TAB>*	-	Moves cursor to the previous input ITEM.
<DELETE CHARACTER>	-	Shift left one character the characters from just right of the cursor to the end

	of the field line. The right most character in the field line is replaced with a blank.
<INSERT CHARACTER>-	Shift right one character the character from the cursor position to the end of the field line. The right most character in the field is lost and the character at the cursor position is replaced with a blank.

3.2.2.1.1 Scrolling, Paging, First and Last Page

The scroll and page keys refer to arrays and not an individual ITEM. See the Terminal Operator's Guide for their use in applications.

3.2.2.1.2 ITEM Insert Line

<INSERT LINE> -	Move down one line the line containing the cursor and all following lines in the ITEM. The last line becomes a saved line. A saved line is a line pushed off the bottom of an ITEM. They are not part of the data available to the FP or applications and are kept only as long as the FP is in the text edit mode and the cursor remains in the same ITEM. They would be used in the following manner: user inserts a line to add some data which does not fill the entire line, then deletes the entire item and pastes it back with fill reformatting the item with the new data. The line with the cursor is filled with blanks.

There are no error conditions with this function.

3.2.2.1.2.1 Input

Current cursor position.
Field pointer.

3.2.2.1.3 ITEM Midline Break

<MIDLINE BREAK> -	All characters from the cursor position to the end of the line are removed and

inserted at the beginning of a newline within the ITEM immediately following the cursor's line. The last line becomes a saved line.

There are no error conditions with this function.

3.2.2.1.3.1 Input

Current cursor position.
Field pointer.

3.2.2.1.3.2 Output

Cursor position.

3.2.2.1.4 Previous Contents Restore

<PREVIOUS CONTENTS> -	The contents of the ITEM which contains the cursor are replaced with the contents it held before the last application function key was pressed.

3.2.2.1.4.1 Input

Current cursor position.
Field pointer.

3.2.2.1.5 Cut and Paste Scenario for ITEM Editing

This is somewhat different from file editing. A sequence of delete lines or one delete entire ITEM replaces the contents of the cut and paste buffer. The lines may be inserted into an ITEM exactly as they were deleted using <PASTE>. In this case if the original lines were longer, they are truncated; if shorter, they are padded on the right with blanks. If there are more lines in the cut and paste buffer than lines in the receiving ITEM, then the extra lines become part of the target ITEM'S saved lines.

The lines may be reformatted and inserted into an ITEM using <FILL>. The lines are reformatted so that as many whole words as possible fit on each line of the item. If a word is too wide to fit on a line of the receiving ITEM, it is wrapped. If a line inserted by a fill does not completely fill a line of the ITEM, it is padded on the right with blanks. If there are more words in the cut and paste buffer than will fit in the ITEM, the extra become part of the target ITEM'S saved lines.

3.2.2.1.5.1 ITEM Delete Line

<DELETE LINE>	- The characters from the cursor position to the end of the line in the ITEM are deleted. If the cursor is at the first position of the line in the ITEM, move up one line the lines following the line containing the cursor and fill in the last line in the field with a saved line or if none are left, with blanks. (see insert line for a description of saved lines). Else the cursor is moved to the first position of the line in the ITEM following the current line. If the user has not moved the cursor since the last <DELETE LINE> the characters are concatenated to the cut and paste buffer. Else they replace it.

There are no error conditions with this function.

3.2.2.1.5.1.1 Input

Current cursor position	- This is the position within the ITEM.
Field pointer	- This is a pointer to the ITEM containing the cursor.

3.2.2.1.5.1.2 Output

Cut and paste buffer.

Cursor position	- If the cursor is at the first position of a line in an ITEM, it is not moved; else it is moved to the first position of the line in the ITEM following the current line.

3.2.2.1.5.2 Delete Entire ITEM

<DELETE ENTIRE ITEM>	- Each line of the ITEM containing the cursor and the ITEM'S saved lines are deleted and replaced with blanks. The lines of the ITEM replace the contents of the cut and paste buffer.

There are no error conditions with this function.

3.2.2.1.5.2.1 <u>Input</u>

Current cursor position.
Field pointer.

3.2.2.1.5.2.2 <u>Output</u>

Cut and paste buffer.
Cursor position.

3.2.2.1.5.3 <u>Paste for Cut and Paste</u>

<PASTE>	- This inserts the contents of the cut and paste buffer at the cursor position within an ITEM.

There are no error conditions with this function.

3.2.2.1.5.3.1 <u>Input</u>

Current cursor position.
Field pointer.
Cut and paste buffer.

3.2.2.1.5.4 <u>Fill for Cut and Paste</u>

<FILL>	- This formats the contents of the cut and paste buffer so that as many whole words as possible can be inserted on each line of the ITEM containing the cursor. Fill margins may be set for an ITEM and are in effect for the fill that follows as long as the cursor remains in the ITEM in which the margins were set.

There are no error conditions with this function.

3.2.2.1.5.4.1 <u>Input</u>

Current cursor position.
Field pointer.
Cut and paste buffer.

3.2.2.1.6 Item Search

<SEARCH>	- The ITEM containing the cursor is searched for the first occurrence of the string after the cursor, and the cursor is positioned at this occurrence. If the command is appended with a minus "-", the search is done "backwards". If the string is not found in the ITEM, a message is displayed, and the cursor position is unchanged.

The following prompts are displayed:

Search String: Direction

3.2.2.1.6.1 Input

Current cursor position.
Field pointer.

string	- The string to search the ITEM for.
Direction	- Blank: forward, minus: backward.

3.2.2.1.6.2 Output

Cursor position.

3.2.2.1.7 ITEM Search Next

<SEARCH NEXT>	- This is equivalent to issuing the SEARCH command with no operands; that is, it repeats the last search.

The error conditions are the same as for the SEARCH command.

3.2.2.1.7.1 Input

Current Cursor Position
Field Pointer

String	- The last string used in a search.
Direction	- The last direction used in a search.

3.2.2.1.7.2 Output

Cursor Position.

3.2.2.1.8 Item Replace

<REPLACE>	- Immediately after a search or search next, a replace may be done to replace the just searched for string with the replacement string.

The following prompts are displayed:

Replacement String: Direction:

3.2.2.1.8.1 Input

Current cursor position.
Field pointer.

String -	The text string replacing the search string.
Direction -	Blank: forward, minus: backwards.

3.2.2.1.8.2 Output

Cursor position.

3.2.2.1.9 Replace Next

<REPLACE NEXT> -	May be used immediately after a search. It repeats the most recently executed replace.

The error conditions are the same as for the <REPLACE> function.

3.2.2.1.9.1 Input

Current Cursor Position.
Field Pointer.
String.
Direction.

3.2.2.1.9.2 Output

Cursor Position

3.2.2.1.10 ITEM - Set limits of the fill margins.

<MARGINS> -	Sets the fill margins for the ITEM containing the cursor. The margins are

in effect for as long as the cursor is within the same ITEM.

The following prompts are displayed:

Fill Margins - Left: Right:

3.2.2.1.10.1 Input

left margin.
right margin.

3.2.2.1.11 Item Repeat

<REPEAT>	- The next command or function key is executed number times within the ITEM containing the cursor. The command is similar to the file editing version. The following functions are supported: <INSERT LINE> <MIDLINE BREAK>, <PASTE>, <FILL>, <DELETE LINE>, and <QUIT>.

The number must be greater than zero. If not, the command is aborted and a warning message is issued.

The following prompt is displayed:

Repeat count:

3.2.2.1.11.1 Input

Current cursor position.
Field pointer.

number	- The number of times the command or function is the be executed.
function key	- This is the item to be repeated.

3.2.2.1.11.2 Output

This is dependent on the command or function repeated.

SECTION 4

QUALITY ASSURANCE PROVISIONS

4.1 Introduction and Definitions

"Testing" is a systematic process that may be preplanned and explicitly stated. Test techniques and procedures may be defined in advance and a sequence of test steps may be specified. "Debugging" is the process of isolation and correction of the cause of an error.

4.2 Computer Programming Test and Evaluation

The quality assurance provisions for test consists of the normal testing techniques that are accomplished during the construction process. They consist of design and code walk-throughs, unit testing, and integration testing. These tests will be performed by the design team.

The integration test developed for the TE consists of a list of commands (and their expected outputs) which are used to create a script output of a test session and its output. This session tests each function and command to ensure its correct operation. Results of the session may be compared with those of the unit testing.

Because the TE consists of a flat hierarchy of modules, unit testing primarily involves testing each of the TE interface routines and internal functions for correct processing and output.

Below the level of modules implementing each function/command is a small set of procedures for assuring the integrity of the buffer and the top of file, bottom of file, cursor position and display start line. Since the TE is actually an FP application, each reference to an FP procedure is verified that it contains the correct parameters and each form is displayed to ensure it is correct.

The test developed for the FPTE is similar to the TE.

SECTION 5

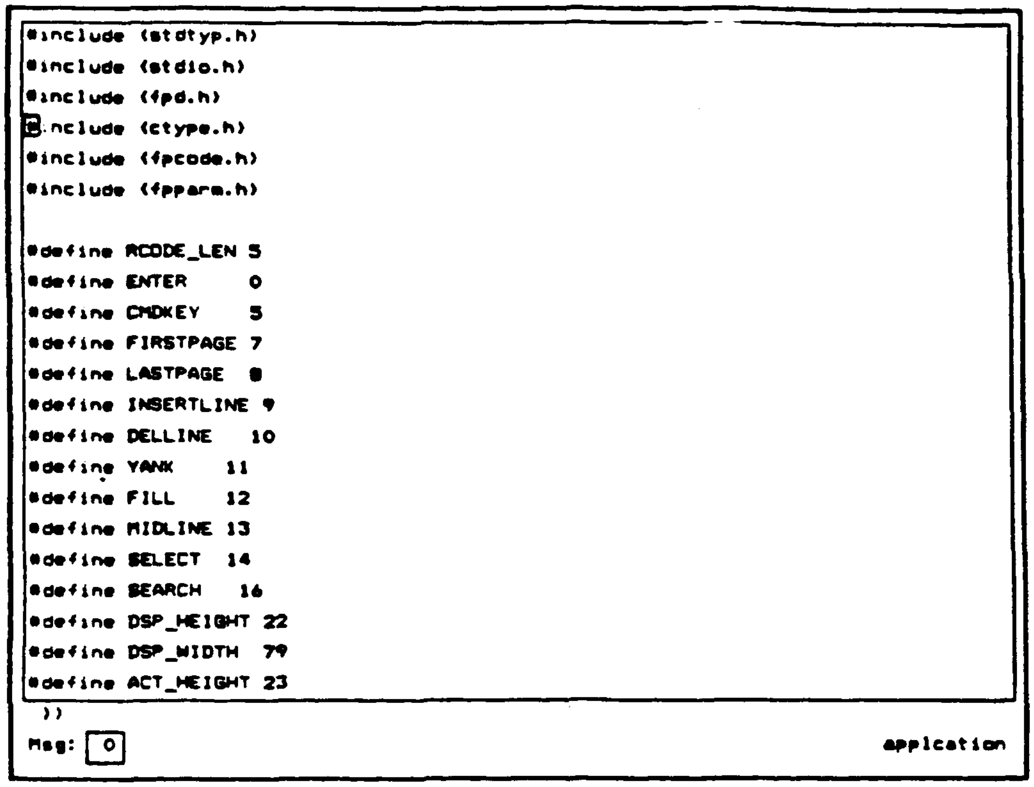
PREPARATION FOR DELIVERY

The implementation site for the constructed software is the Integrated Support System (IISS) Test Bed site located at the General Electric Company, in Albany, NY. The Software associated with each TE release is clearly identified and includes instructions on procedures to be followed for installation of the release.

APPENDIX A

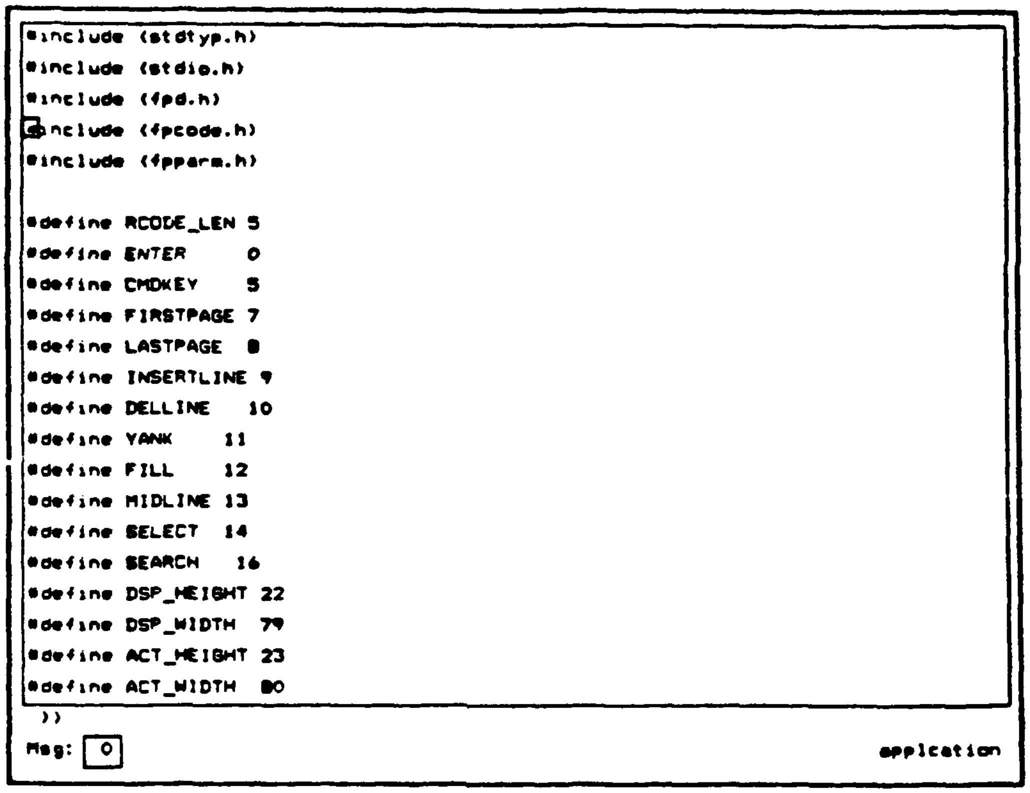
FIGURES OF BEFORE AND AFTER AN OPERATION

In all figures the designation in parenthesis refers to the paragraph in Section 3. The cursor is indicated by a box.



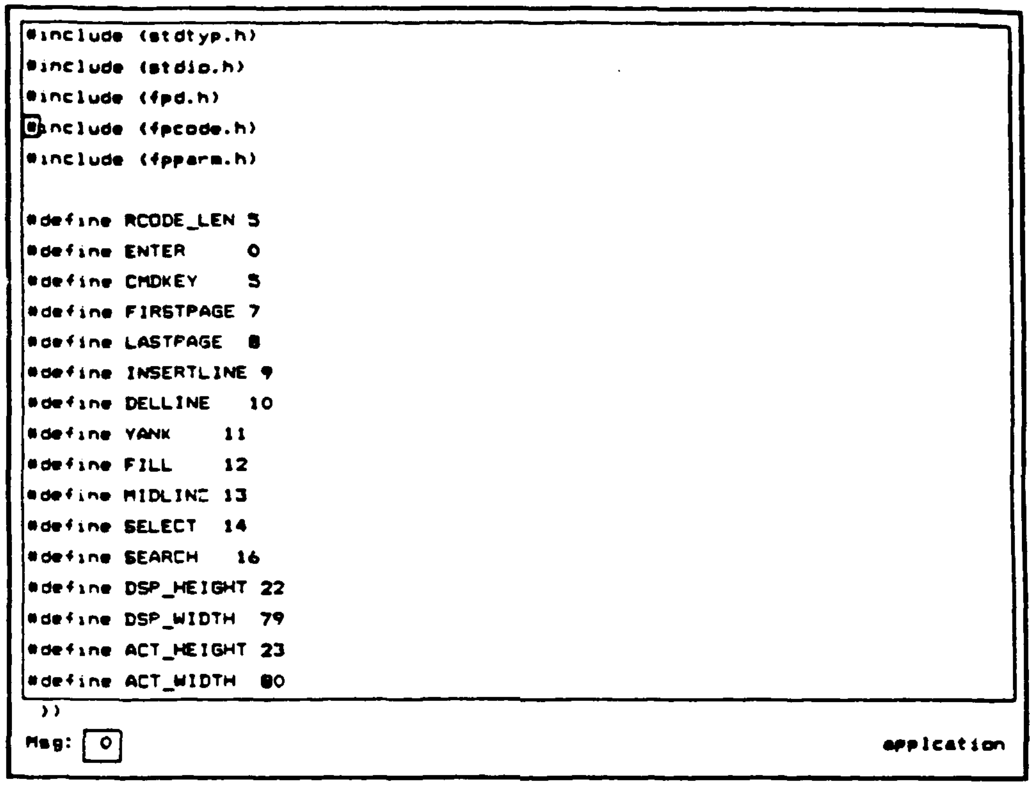
```
#include <stdtyp.h>
#include <stdio.h>
#include <fpd.h>
#include <ctype.h>
#include <fpcode.h>
#include <fpparm.h>

#define RCODE_LEN 5
#define ENTER      0
#define CMDKEY     5
#define FIRSTPAGE 7
#define LASTPAGE  8
#define INSERTLINE 9
#define DELLINE    10
#define YANK     11
#define FILL     12
#define MIDLINE 13
#define SELECT   14
#define SEARCH    16
#define DSP_HEIGHT 22
#define DSP_WIDTH  79
#define ACT_HEIGHT 23
 ))
Msg: 0                                                  applcation
```

Figure A-1 Screen Before Delete Line (3.2.1.1.10.2.a)



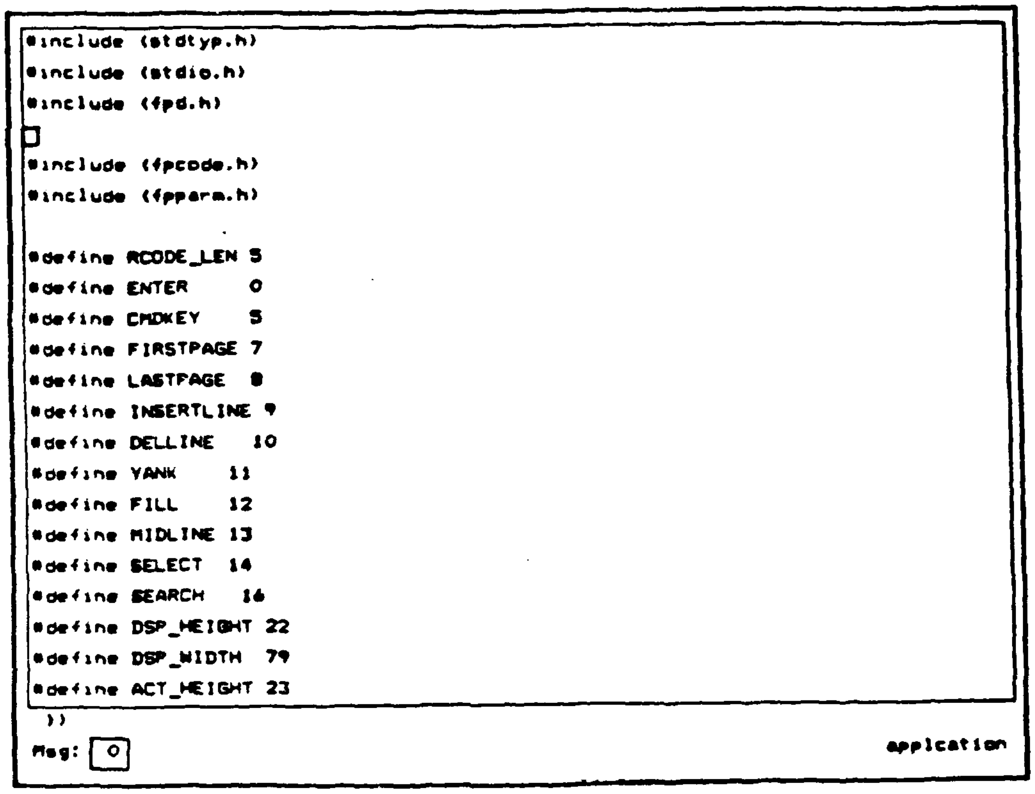
```
#include <stdtyp.h>
#include <stdio.h>
#include <fpd.h>
#include <fpcode.h>
#include <fpparm.h>

#define RCODE_LEN 5
#define ENTER      0
#define CMDKEY     5
#define FIRSTPAGE 7
#define LASTPAGE  8
#define INSERTLINE 9
#define DELLINE   10
#define YANK      11
#define FILL      12
#define MIDLINE 13
#define SELECT  14
#define SEARCH    16
#define DSP_HEIGHT 22
#define DSP_WIDTH  79
#define ACT_HEIGHT 23
#define ACT_WIDTH  80
 >>
Msg: 0                                              applcation
```

Figure A-2 Screen After Delete Line (3.2.1.1.10.2.b)



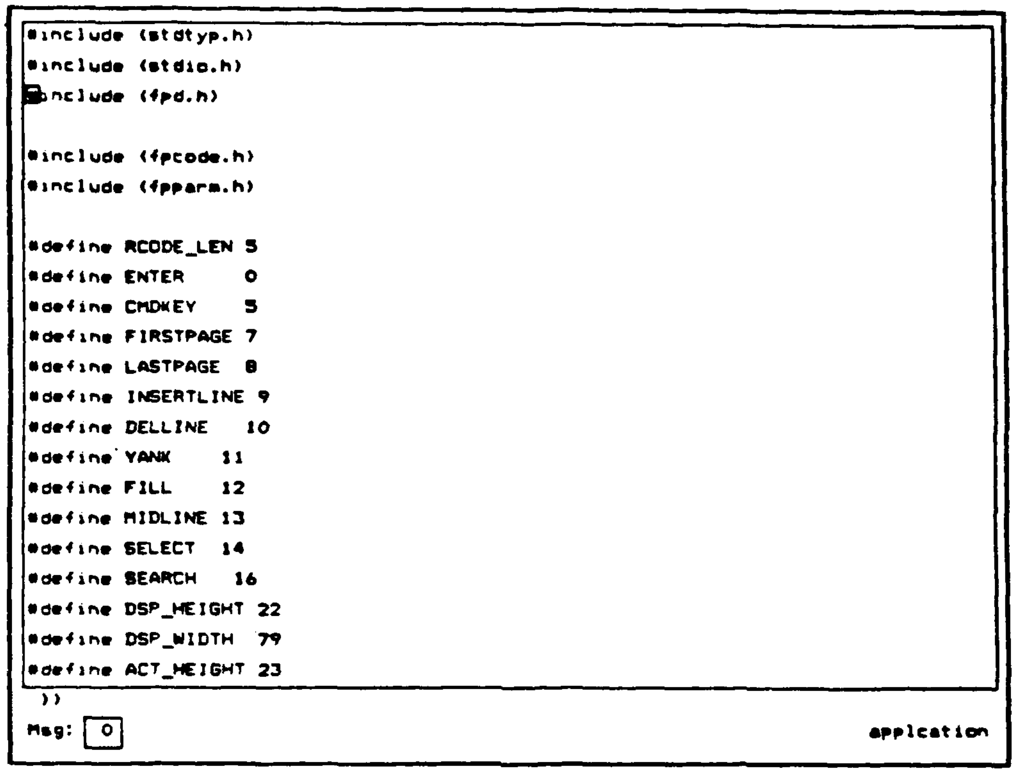
```
#include <stdtyp.h>
#include <stdio.h>
#include <fpd.h>
#include <fpcode.h>
#include <fpparm.h>

#define RCODE_LEN 5
#define ENTER     0
#define CMDKEY    5
#define FIRSTPAGE 7
#define LASTPAGE  8
#define INSERTLINE 9
#define DELLINE   10
#define YANK     11
#define FILL     12
#define MIDLINE 13
#define SELECT  14
#define SEARCH   16
#define DSP_HEIGHT 22
#define DSP_WIDTH  79
#define ACT_HEIGHT 23
#define ACT_WIDTH  80
 ))
Msg: 0                                            application
```

Figure A-3 Screen Before Insert Line (3.2.1.1.8.a)



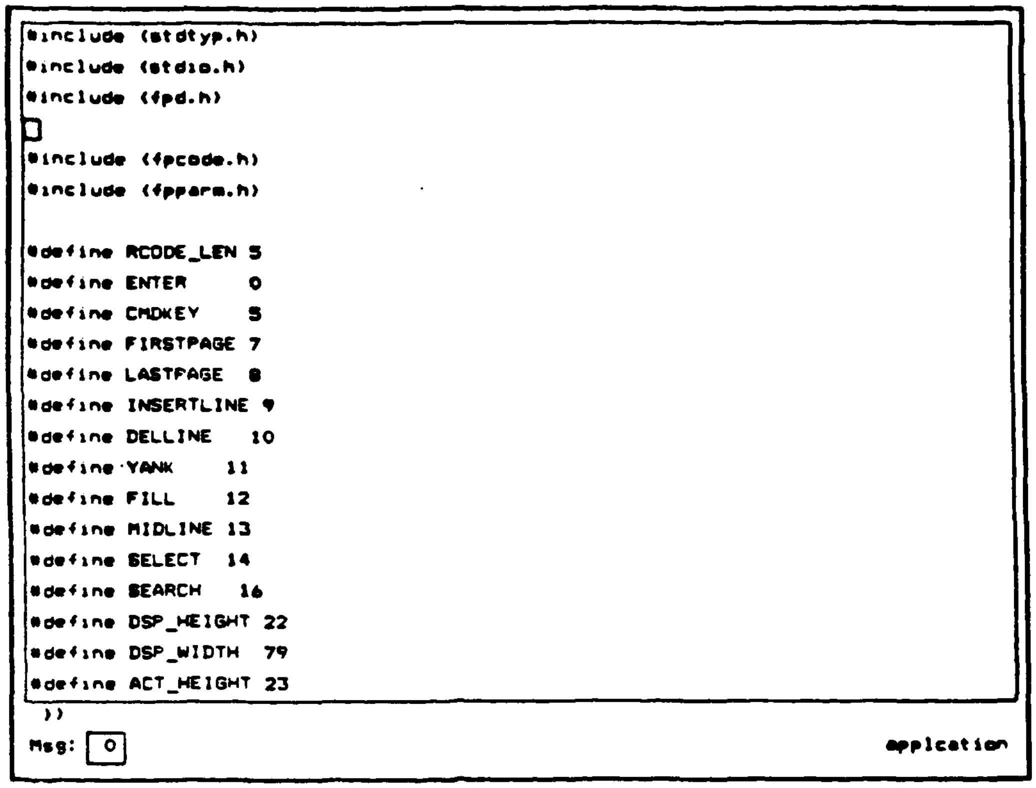
```
#include (stdtyp.h)
#include (stdio.h)
#include (fpd.h)
□
#include (fpcode.h)
#include (fpparm.h)

#define RCODE_LEN 5
#define ENTER      0
#define CMDKEY     5
#define FIRSTPAGE 7
#define LASTPAGE  8
#define INSERTLINE 9
#define DELLINE    10
#define YANK      11
#define FILL      12
#define MIDLINE 13
#define SELECT  14
#define SEARCH    16
#define DSP_HEIGHT 22
#define DSP_WIDTH  79
#define ACT_HEIGHT 23
 ))
Msg: [0]                                                        application
```

Figure A-4 After Insert Line (3.2.1.1.8.b)



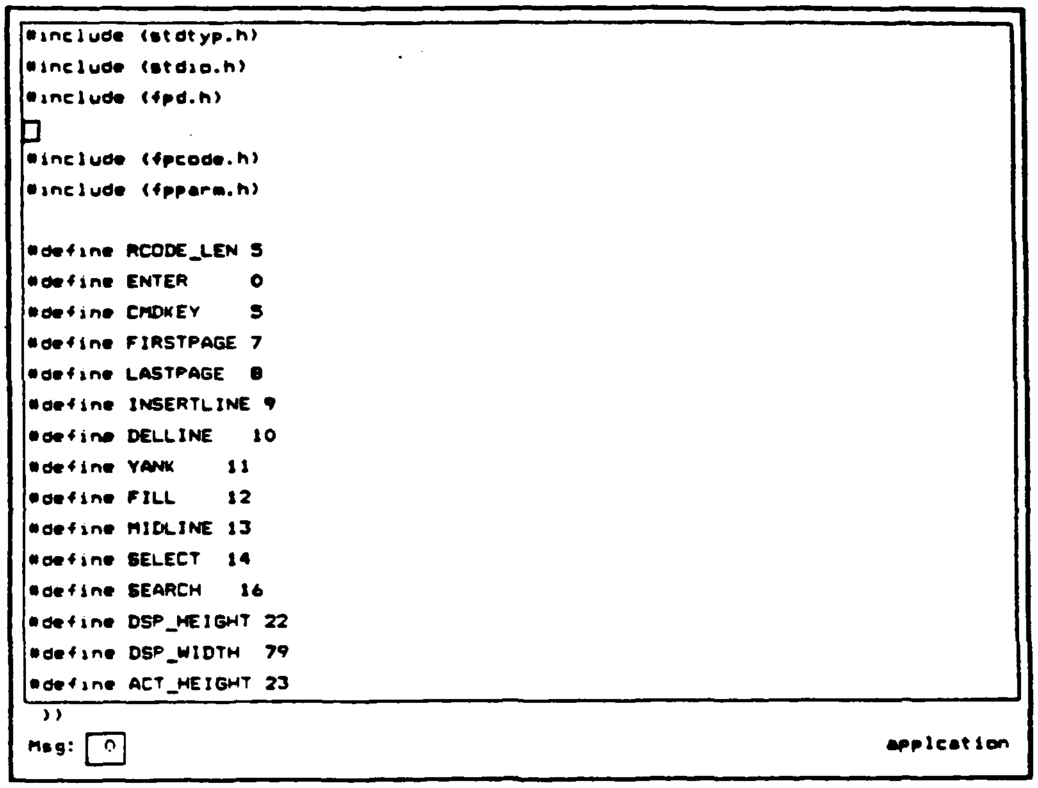
```
#include <stdtyp.h>
#include <stdio.h>
#include <fpd.h>

#include <fpcode.h>
#include <fpparm.h>

#define RCODE_LEN 5
#define ENTER     0
#define CMDKEY    5
#define FIRSTPAGE 7
#define LASTPAGE  8
#define INSERTLINE 9
#define DELLINE   10
#define YANK     11
#define FILL     12
#define MIDLINE 13
#define SELECT  14
#define SEARCH   16
#define DSP_HEIGHT 22
#define DSP_WIDTH  79
#define ACT_HEIGHT 23
 >>
Msg: [0]                                             applcation
```

Figure A-5 Before Next Line Command - Scroll up (3.2.1.1.2.a)



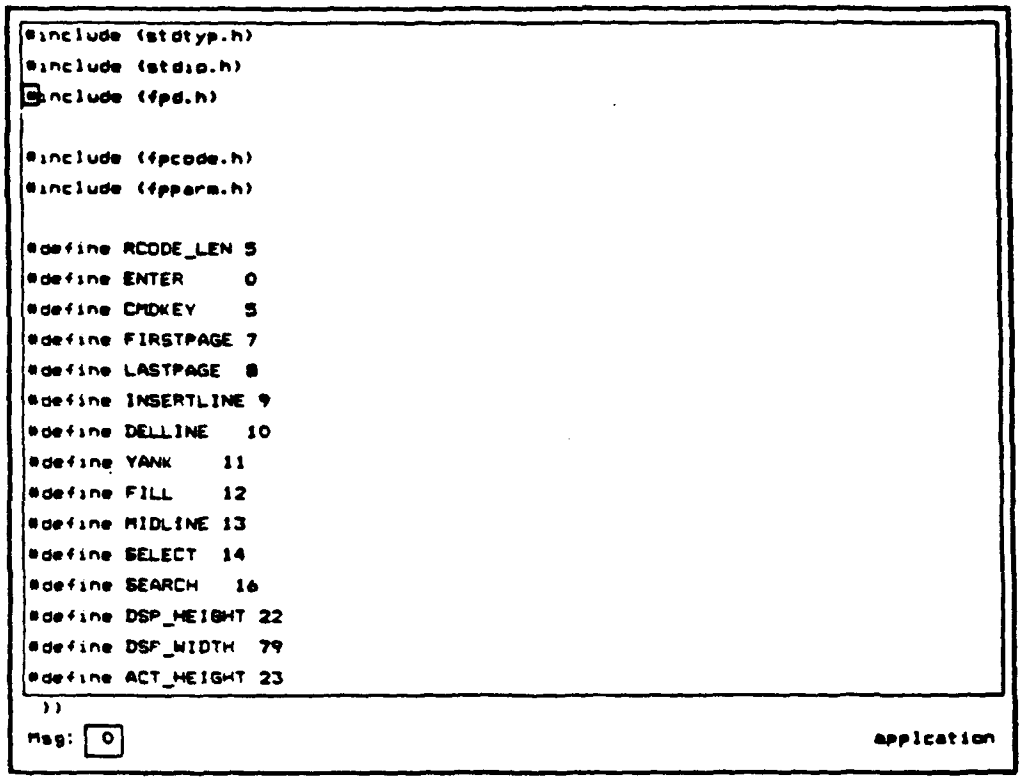
```
#include <stdtyp.h>
#include <stdio.h>
#include <fpd.h>

#include <fpcode.h>
#include <fpparm.h>

#define RCODE_LEN 5
#define ENTER     0
#define CMDKEY    5
#define FIRSTPAGE 7
#define LASTPAGE  8
#define INSERTLINE 9
#define DELLINE   10
#define YANK      11
#define FILL      12
#define MIDLINE 13
#define SELECT  14
#define SEARCH   16
#define DSP_HEIGHT 22
#define DSP_WIDTH  79
#define ACT_HEIGHT 23
 >>
Msg: 0                                                          application
```

Figure A-6 After Next Line Command - Scroll up (3.2.1.1.2.b)



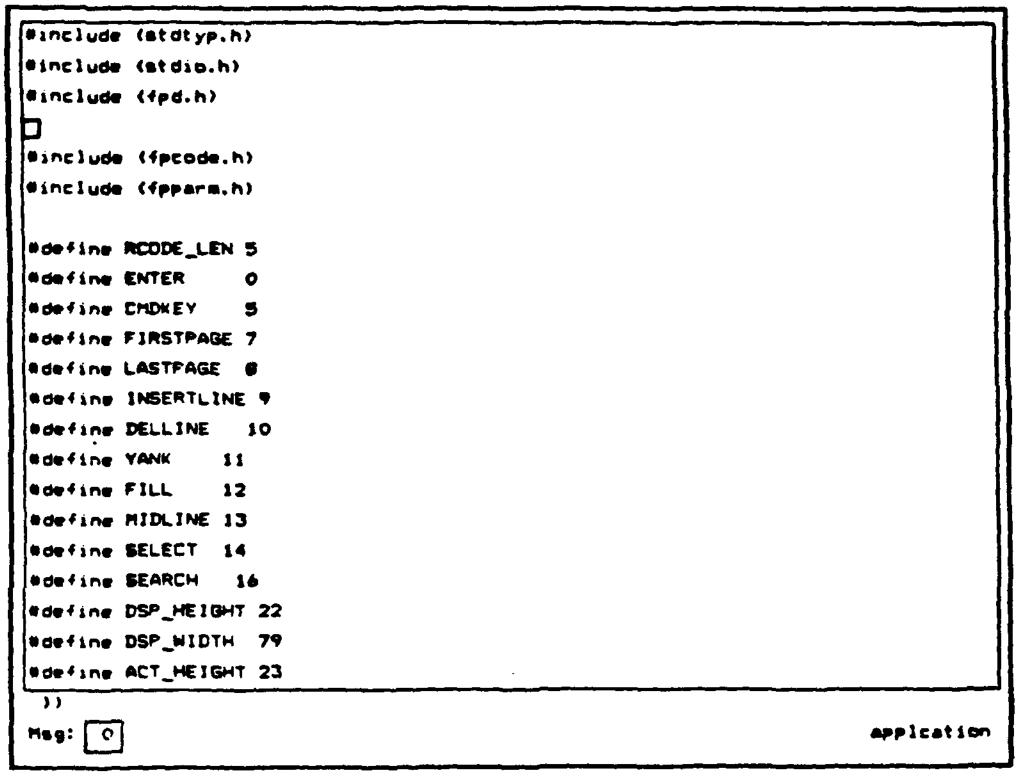
```
#include (stdtyp.h)
#include (stdio.h)
#include (fpd.h)

#include (fpcode.h)
#include (fpparm.h)

#define RCODE_LEN 5
#define ENTER     0
#define CMDKEY    5
#define FIRSTPAGE 7
#define LASTPAGE  8
#define INSERTLINE 9
#define DELLINE   10
#define YANK      11
#define FILL      12
#define MIDLINE 13
#define SELECT  14
#define SEARCH   16
#define DSP_HEIGHT 22
#define DSP_WIDTH  79
#define ACT_HEIGHT 23
 ))
Msg: 0                                              applcation
```

Figure A-7 Before Previous Line Command - Scroll down (3.2.1.1.3.a)



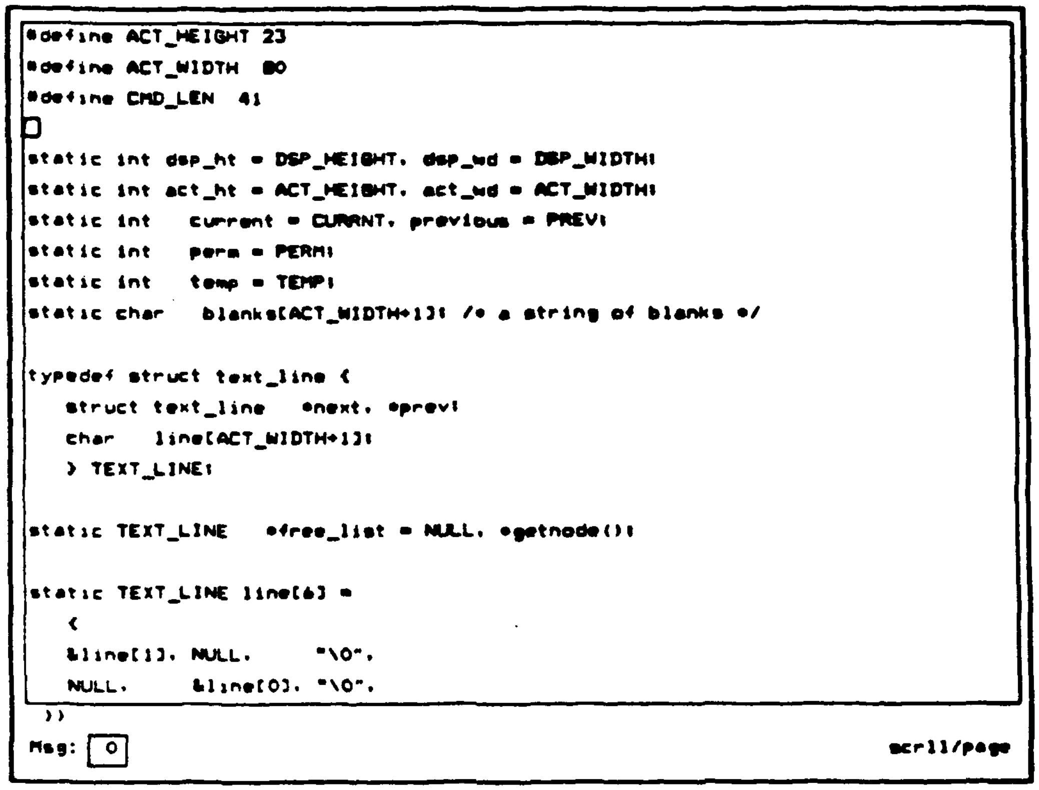
```
#include <stdtyp.h>
#include <stdio.h>
#include <fpd.h>

#include <fpcode.h>
#include <fpparm.h>

#define RCODE_LEN 5
#define ENTER      0
#define CMDKEY     5
#define FIRSTPAGE  7
#define LASTPAGE   8
#define INSERTLINE 9
#define DELLINE    10
#define YANK       11
#define FILL       12
#define MIDLINE    13
#define SELECT     14
#define SEARCH     16
#define DSP_HEIGHT 22
#define DSP_WIDTH  79
#define ACT_HEIGHT 23
 ))
Msg: 0                                                  application
```

Figure A-8 After Previous Line Command - Scroll down (3.2.1.1.3.b)



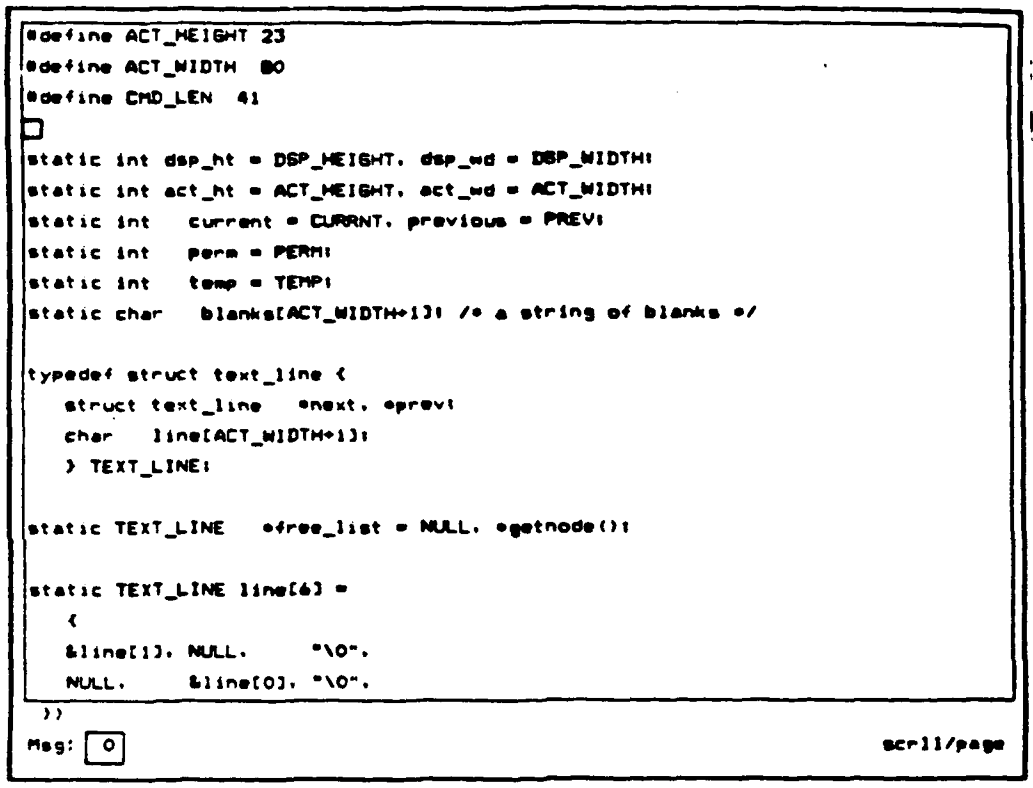
```
#include (stdtyp.h)
#include (stdio.h)
#include (fpd.h)

#include (fpcode.h)
#include (fpparm.h)

#define RCODE_LEN 5
#define ENTER      0
#define CMDKEY     5
#define FIRSTPAGE 7
#define LASTPAGE  8
#define INSERTLINE 9
#define DELLINE   10
#define YANK      11
#define FILL      12
#define MIDLINE 13
#define SELECT  14
#define SEARCH   16
#define DSP_HEIGHT 22
#define DSP_WIDTH  79
#define ACT_HEIGHT 23
 ))
Msg: 0                                    applcation
```

Figure A-9 Before Next Page - page up (3.2.1.1.4.a)



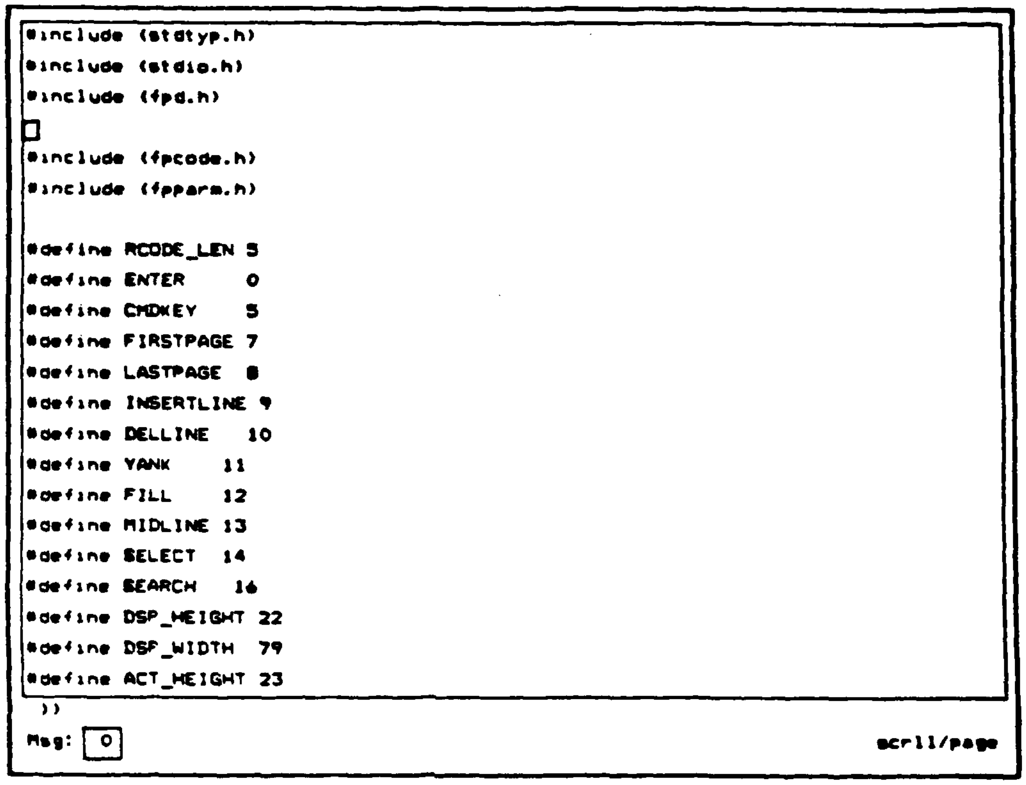
```
#define ACT_HEIGHT 23
#define ACT_WIDTH  80
#define CMD_LEN  41

static int dsp_ht = DSP_HEIGHT, dsp_wd = DSP_WIDTH;
static int act_ht = ACT_HEIGHT, act_wd = ACT_WIDTH;
static int   current = CURRNT, previous = PREV;
static int   perm = PERM;
static int   temp = TEMP;
static char   blanks[ACT_WIDTH+1]; /* a string of blanks */

typedef struct text_line {
   struct text_line   *next, *prev;
   char   line[ACT_WIDTH+1];
   } TEXT_LINE;

static TEXT_LINE   *free_list = NULL, *getnode();

static TEXT_LINE line[6] =
   {
   &line[1], NULL,     "\0",
   NULL,     &line[0], "\0",
 }}
Msg: 0                                                  scrll/page
```

Figure A-10 After Next Page - page up (3.2.1.1.4.b)



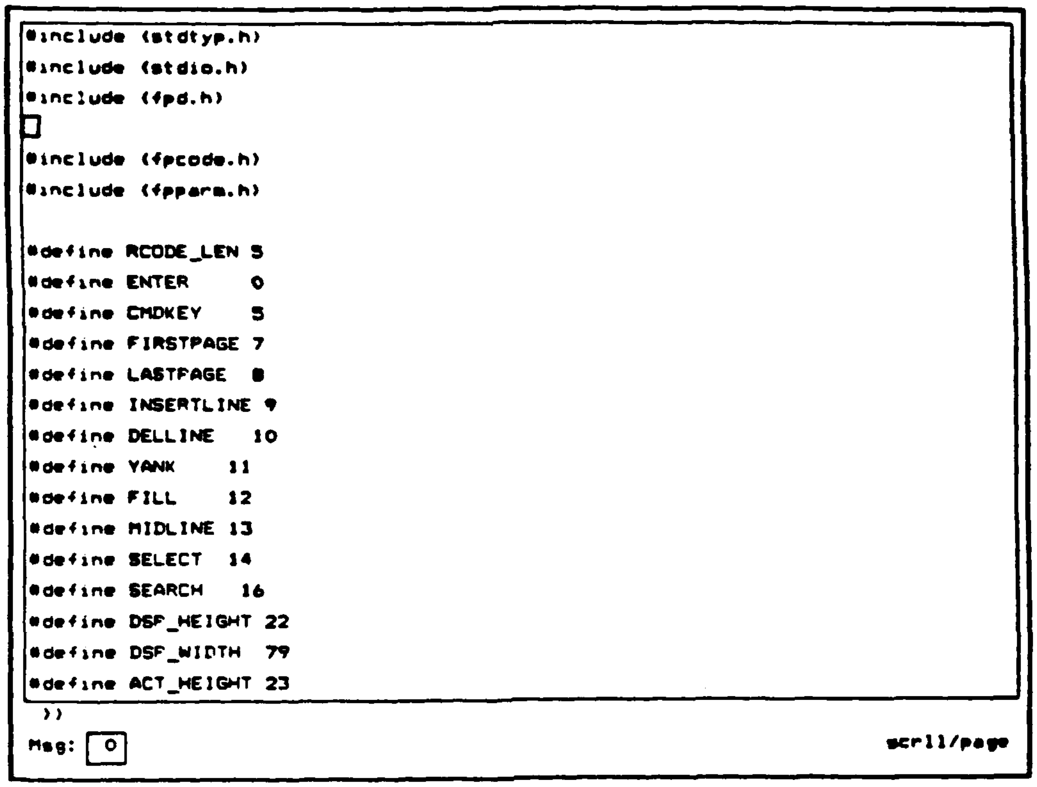
```
#define ACT_HEIGHT 23
#define ACT_WIDTH  80
#define CMD_LEN  41

static int dsp_ht = DSP_HEIGHT, dsp_wd = DSP_WIDTH;
static int act_ht = ACT_HEIGHT, act_wd = ACT_WIDTH;
static int   current = CURRNT, previous = PREV;
static int   perm = PERM;
static int   temp = TEMP;
static char   blanks[ACT_WIDTH+1]; /* a string of blanks */

typedef struct text_line {
   struct text_line   *next, *prev;
   char   line[ACT_WIDTH+1];
   } TEXT_LINE;

static TEXT_LINE   *free_list = NULL, *getnode();

static TEXT_LINE line[6] =
   {
   &line[1], NULL,     "\0",
   NULL,     &line[0], "\0",
 }}
Msg: 0                                                   scrll/page
```

Figure A-11 Before Previous Page - page down (3.2.1.1.5.a)



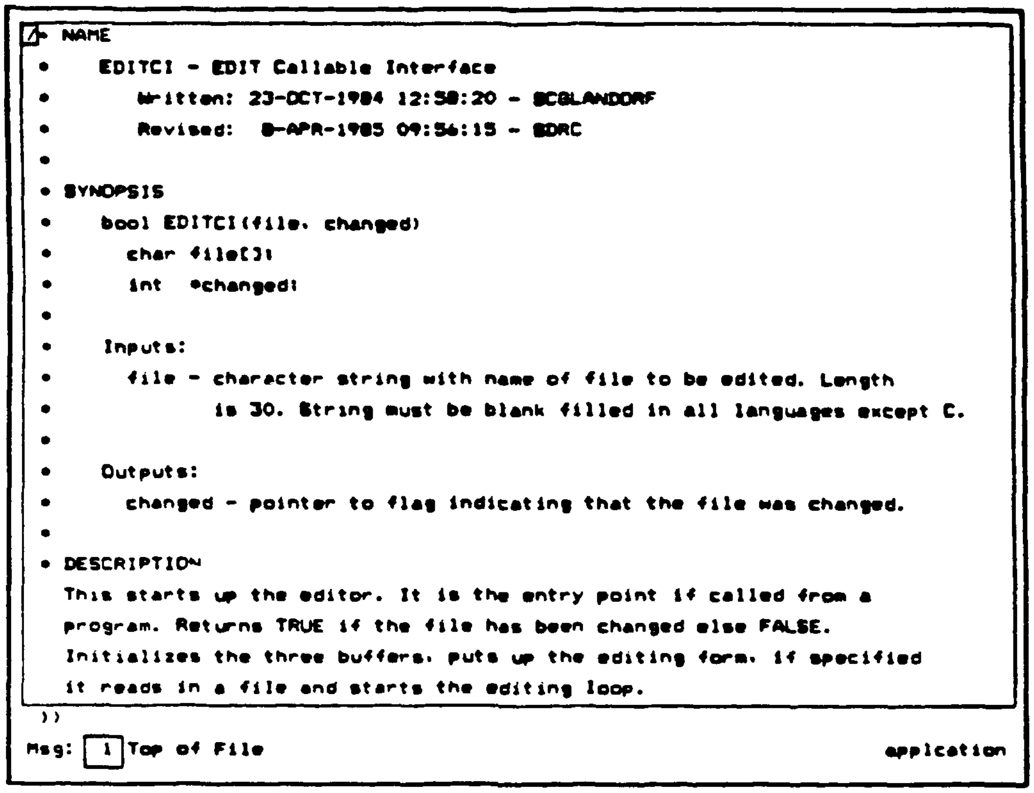
```
#include (stdtyp.h)
#include (stdio.h)
#include (fpd.h)
□
#include (fpcode.h)
#include (fpparm.h)

#define RCODE_LEN 5
#define ENTER     0
#define CMDKEY    5
#define FIRSTPAGE 7
#define LASTPAGE  8
#define INSERTLINE 9
#define DELLINE   10
#define YANK     11
#define FILL     12
#define MIDLINE 13
#define SELECT  14
#define SEARCH   16
#define DSP_HEIGHT 22
#define DSP_WIDTH  79
#define ACT_HEIGHT 23
 ))
Msg: 0                                              scrll/page
```

Figure A-12 After Previous Page - page down (3.2.1.1.5.b)



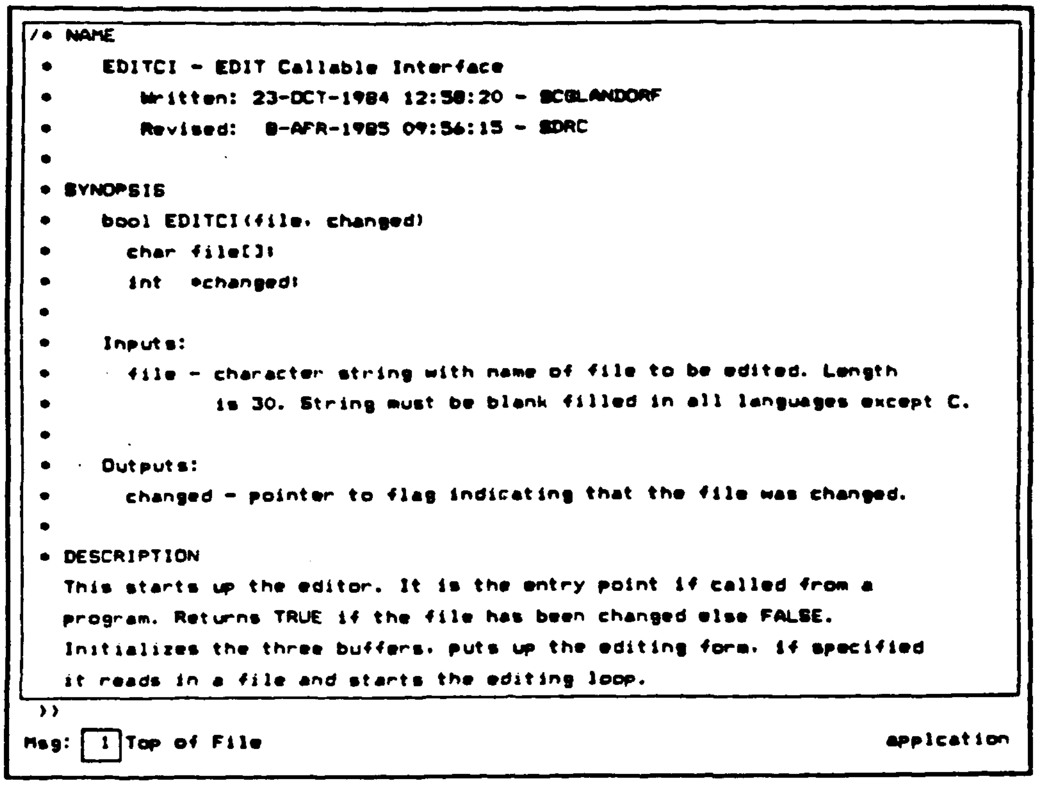
```
#include <stdtyp.h>
#include <stdio.h>
#include <fpd.h>
□
#include <fpcode.h>
#include <fpparm.h>

#define RCODE_LEN 5
#define ENTER      0
#define CMDKEY     5
#define FIRSTPAGE 7
#define LASTPAGE  8
#define INSERTLINE 9
#define DELLINE    10
#define YANK      11
#define FILL      12
#define MIDLINE 13
#define SELECT   14
#define SEARCH    16
#define DSP_HEIGHT 22
#define DSP_WIDTH  79
#define ACT_HEIGHT 23
 >>
Msg: [0]                                                      scrll/page
```

Figure A-13 Before First Page (3.2.1.1.6.a)



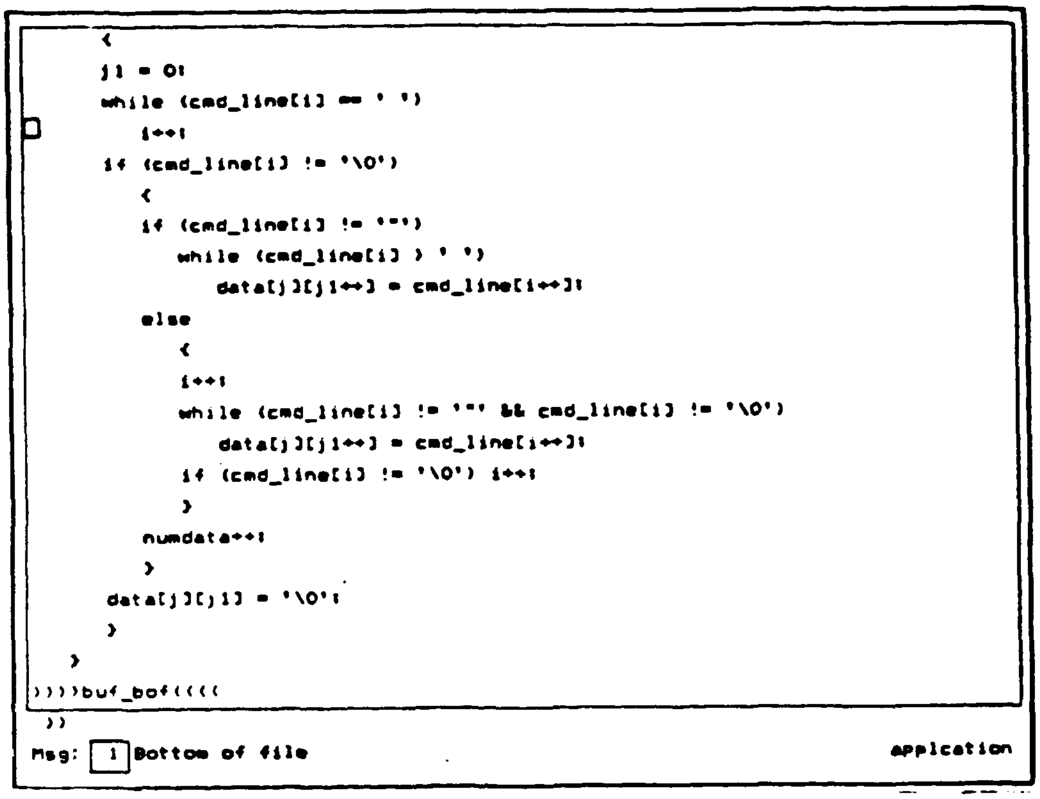
```
/* NAME
 *    EDITCI - EDIT Callable Interface
 *       Written: 23-OCT-1984 12:58:20 - BCBLANDORF
 *       Revised:  8-APR-1985 09:56:15 - BDRC
 *
 * SYNOPSIS
 *    bool EDITCI(file, changed)
 *      char file[];
 *      int  *changed;
 *
 *    Inputs:
 *      file - character string with name of file to be edited. Length
 *             is 30. String must be blank filled in all languages except C.
 *
 *    Outputs:
 *      changed - pointer to flag indicating that the file was changed.
 *
 * DESCRIPTION
   This starts up the editor. It is the entry point if called from a
   program. Returns TRUE if the file has been changed else FALSE.
   Initializes the three buffers, puts up the editing form, if specified
   it reads in a file and starts the editing loop.
 ))
Msg: 1 Top of File                                              application
```

Figure A-14 After First Page (3.2.1.1.6.b)



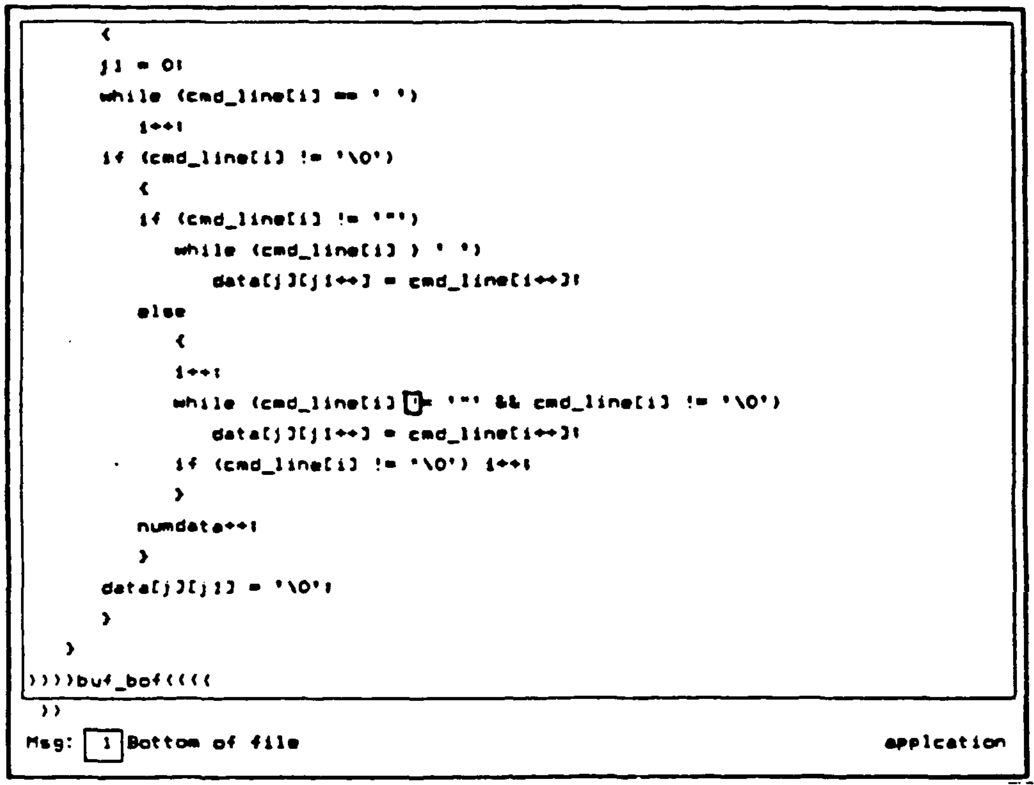
```
/* NAME
 *    EDITCI - EDIT Callable Interface
 *       Written: 23-OCT-1984 12:58:20 - SCGLANDORF
 *       Revised:  8-APR-1985 09:56:15 - SDRC
 *
 * SYNOPSIS
 *    bool EDITCI(file, changed)
 *      char file[];
 *      int  *changed;
 *
 *    Inputs:
 *      file - character string with name of file to be edited. Length
 *             is 30. String must be blank filled in all languages except C.
 *
 *    Outputs:
 *      changed - pointer to flag indicating that the file was changed.
 *
 * DESCRIPTION
   This starts up the editor. It is the entry point if called from a
   program. Returns TRUE if the file has been changed else FALSE.
   Initializes the three buffers, puts up the editing form, if specified
   it reads in a file and starts the editing loop.
 >>
Msg: 1 Top of File                                              applcation
```

Figure A-15 Before Last Page (3.2.1.1.7.a)



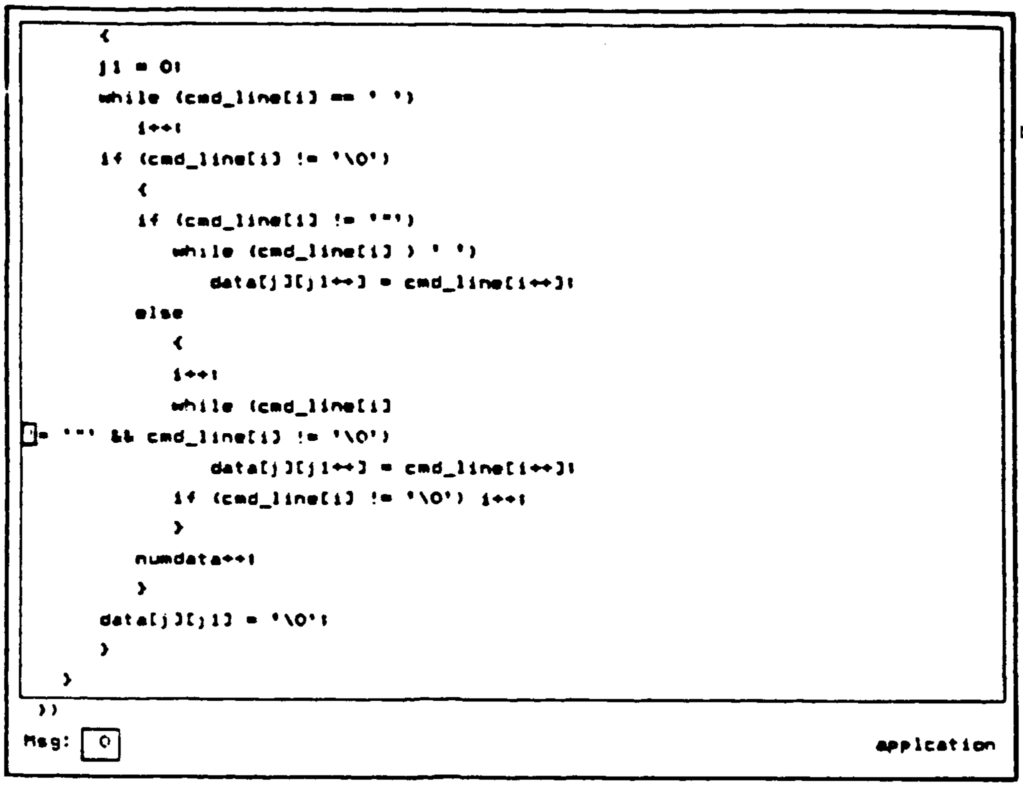
```
        {
        ji = 0;
        while (cmd_line[i] == ' ')
           i++;
        if (cmd_line[i] != '\0')
           {
           if (cmd_line[i] != '"')
              while (cmd_line[i] > ' ')
                 data[j][ji++] = cmd_line[i++];
           else
              {
              i++;
              while (cmd_line[i] != '"' && cmd_line[i] != '\0')
                 data[j][ji++] = cmd_line[i++];
              if (cmd_line[i] != '\0') i++;
              }
           numdata++;
           }
        data[j][ji] = '\0';
        }
     }
))))buf_bof((((
 ))
Msg: 1 Bottom of file                                   application
```

Figure A-16 After Last Page (3.2.1.1.7.b)



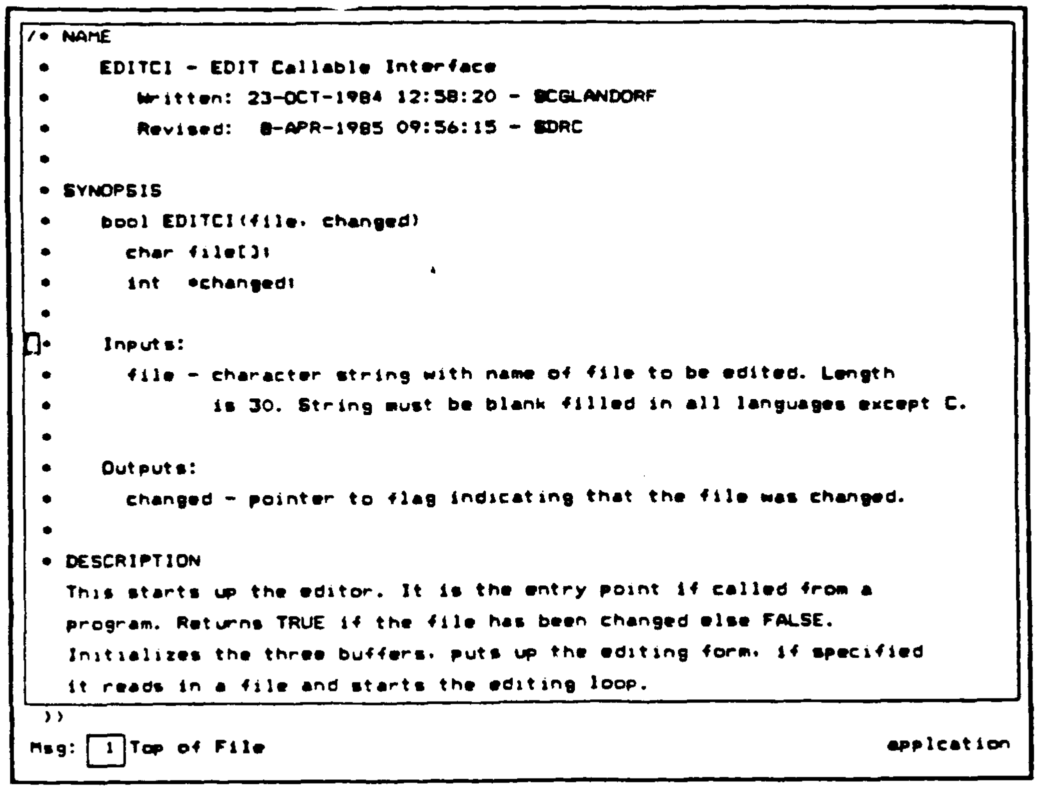
```
      {
      j1 = 0;
      while (cmd_line[i] == ' ')
         i++;
      if (cmd_line[i] != '\0')
         {
         if (cmd_line[i] != '"')
            while (cmd_line[i] > ' ')
               data[j][j1++] = cmd_line[i++];
         else
            {
            i++;
            while (cmd_line[i] != '"' && cmd_line[i] != '\0')
               data[j][j1++] = cmd_line[i++];
            if (cmd_line[i] != '\0') i++;
            }
         numdata++;
         }
      data[j][j1] = '\0';
      }
   }
))))buf_bof((((
 ))
Msg: 1 Bottom of file                                       applcation
```

Figure A-17 Before Midline Break (3.2.1.1.9.a)



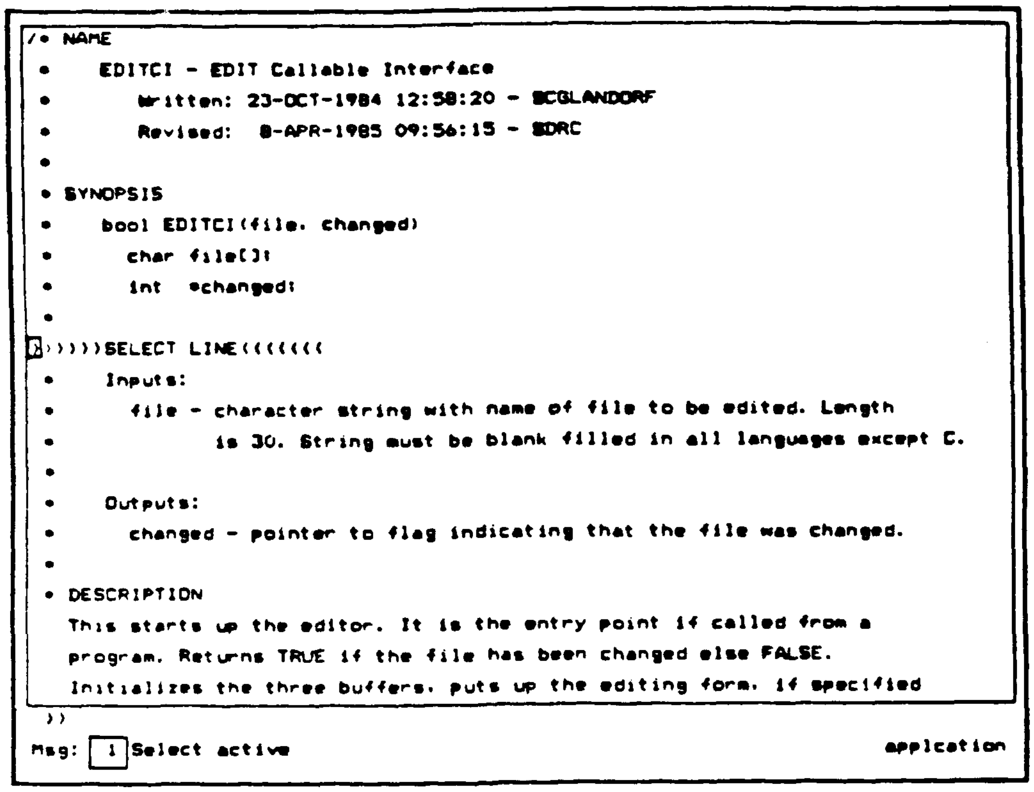
```
        {
        j1 = 0;
        while (cmd_line[i] == ' ')
            i++;
        if (cmd_line[i] != '\0')
            {
            if (cmd_line[i] != '"')
                while (cmd_line[i] > ' ')
                    data[j][j1++] = cmd_line[i++];
            else
                {
                i++;
                while (cmd_line[i]
!= '"' && cmd_line[i] != '\0')
                    data[j][j1++] = cmd_line[i++];
                if (cmd_line[i] != '\0') i++;
                }
            numdata++;
            }
        data[j][j1] = '\0';
        }
    }
 ))
Msg: 0                                        application
```

Figure A-18 After Midline Break (3.2.1.1.9.b)



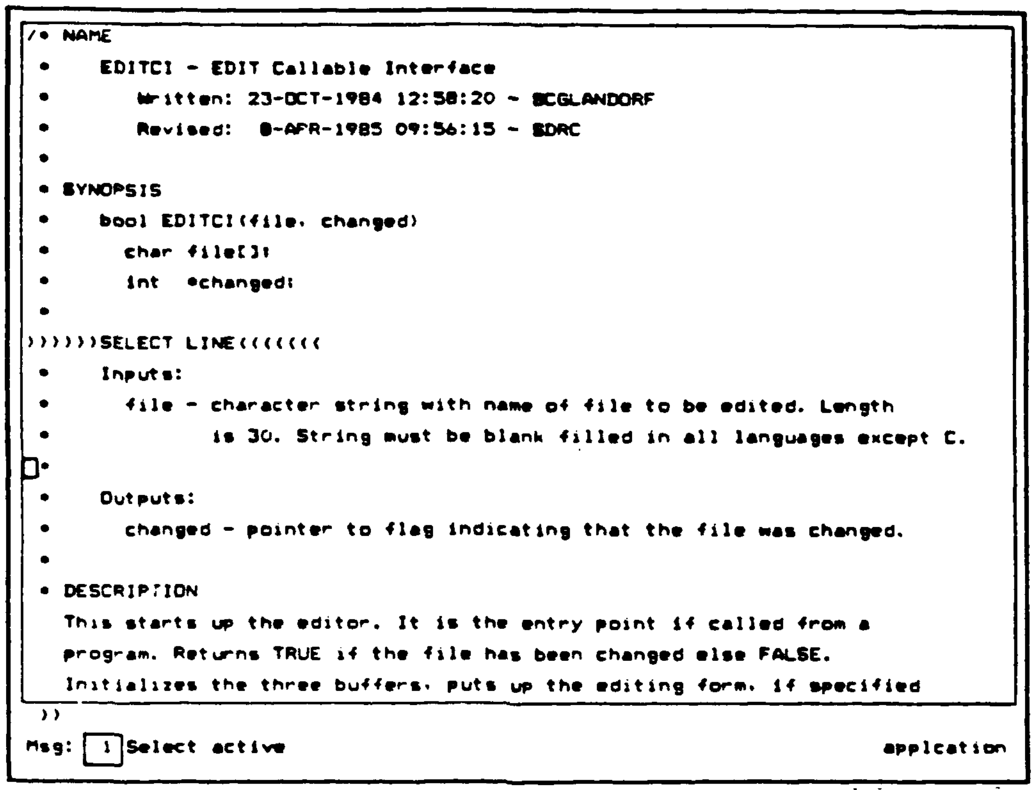
```
/* NAME
 *    EDITCI - EDIT Callable Interface
 *       Written: 23-OCT-1984 12:58:20 - SCGLANDORF
 *       Revised:  8-APR-1985 09:56:15 - SDRC
 *
 * SYNOPSIS
 *    bool EDITCI(file, changed)
 *      char file[];
 *      int  *changed;
 *
 *    Inputs:
 *      file - character string with name of file to be edited. Length
 *             is 30. String must be blank filled in all languages except C.
 *
 *    Outputs:
 *      changed - pointer to flag indicating that the file was changed.
 *
 * DESCRIPTION
   This starts up the editor. It is the entry point if called from a
   program. Returns TRUE if the file has been changed else FALSE.
   Initializes the three buffers, puts up the editing form, if specified
   it reads in a file and starts the editing loop.
 >>
Msg: 1 Top of File                                              applcation
```

Figure A-19 Before Select for Cut and Paste (3.2.1.1.10.a)



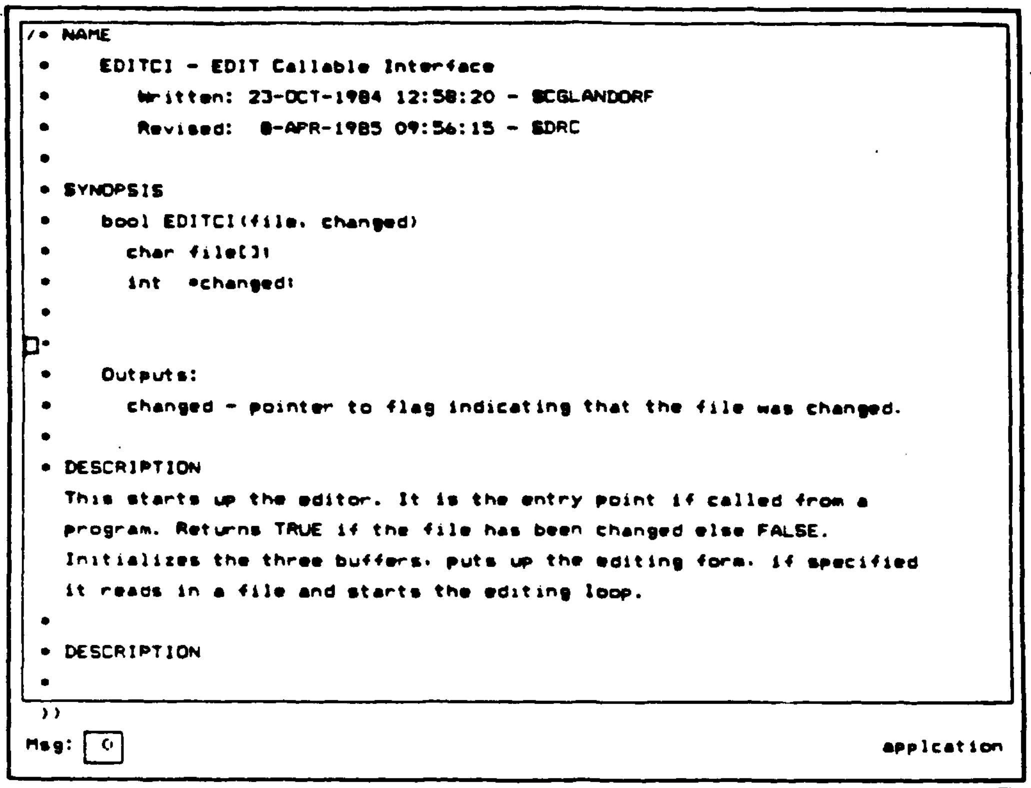
```
/* NAME
 *    EDITCI - EDIT Callable Interface
 *       Written: 23-OCT-1984 12:58:20 - SCGLANDORF
 *       Revised:  8-APR-1985 09:56:15 - SDRC
 *
 * SYNOPSIS
 *    bool EDITCI(file, changed)
 *      char file[];
 *      int  *changed;
 *
>>>>>>SELECT LINE<<<<<<<
 *    Inputs:
 *      file - character string with name of file to be edited. Length
 *             is 30. String must be blank filled in all languages except C.
 *
 *    Outputs:
 *      changed - pointer to flag indicating that the file was changed.
 *
 * DESCRIPTION
   This starts up the editor. It is the entry point if called from a
   program. Returns TRUE if the file has been changed else FALSE.
   Initializes the three buffers, puts up the editing form, if specified
 >>
Msg: 1 Select active                                         applcation
```

Figure A-20 After Select for Cut and Paste (3.2.1.1.10.b)



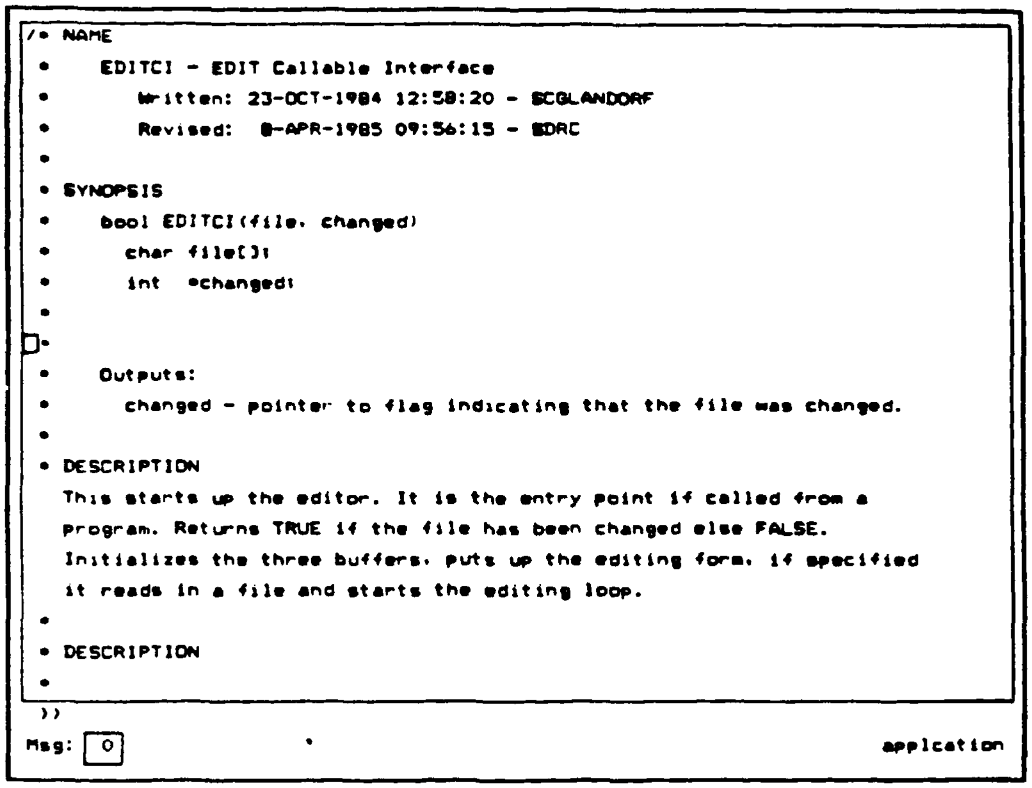
```
/* NAME
 *    EDITCI - EDIT Callable Interface
 *       Written: 23-OCT-1984 12:58:20 - $CGLANDORF
 *       Revised:  8-APR-1985 09:56:15 - $DRC
 *
 * SYNOPSIS
 *    bool EDITCI(file, changed)
 *      char file[];
 *      int  *changed;
 *
))))))SELECT LINE(((((((
 *    Inputs:
 *      file - character string with name of file to be edited. Length
 *             is 30. String must be blank filled in all languages except C.
 *
 *    Outputs:
 *      changed - pointer to flag indicating that the file was changed.
 *
 * DESCRIPTION
   This starts up the editor. It is the entry point if called from a
   program. Returns TRUE if the file has been changed else FALSE.
   Initializes the three buffers, puts up the editing form, if specified
 ))
Msg: [1]Select active                                          application
```

Figure A-21 Positioning Cursor for Delete for Cut and Paste (3.2.1.1.10.c)



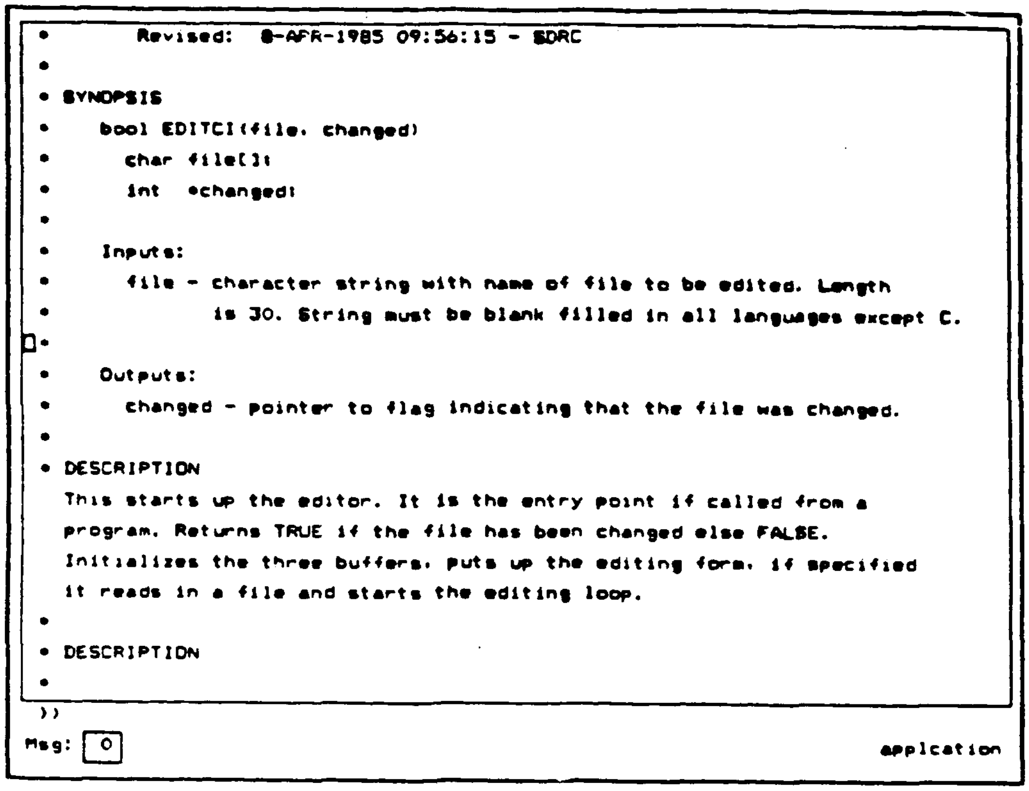
```
/* NAME
 *    EDITCI - EDIT Callable Interface
 *       Written: 23-OCT-1984 12:58:20 - SCGLANDORF
 *       Revised:  8-APR-1985 09:56:15 - SDRC
 *
 * SYNOPSIS
 *    bool EDITCI(file, changed)
 *      char file[];
 *      int  *changed;
 *
 *
 *    Outputs:
 *      changed - pointer to flag indicating that the file was changed.
 *
 * DESCRIPTION
   This starts up the editor. It is the entry point if called from a
   program. Returns TRUE if the file has been changed else FALSE.
   Initializes the three buffers, puts up the editing form, if specified
   it reads in a file and starts the editing loop.
 *
 * DESCRIPTION
 *
 ))
Msg: 0                                                          applcation
```

Figure A-22 After Hitting Delete Line for Cut and Paste (3.2.1.1.10.d)



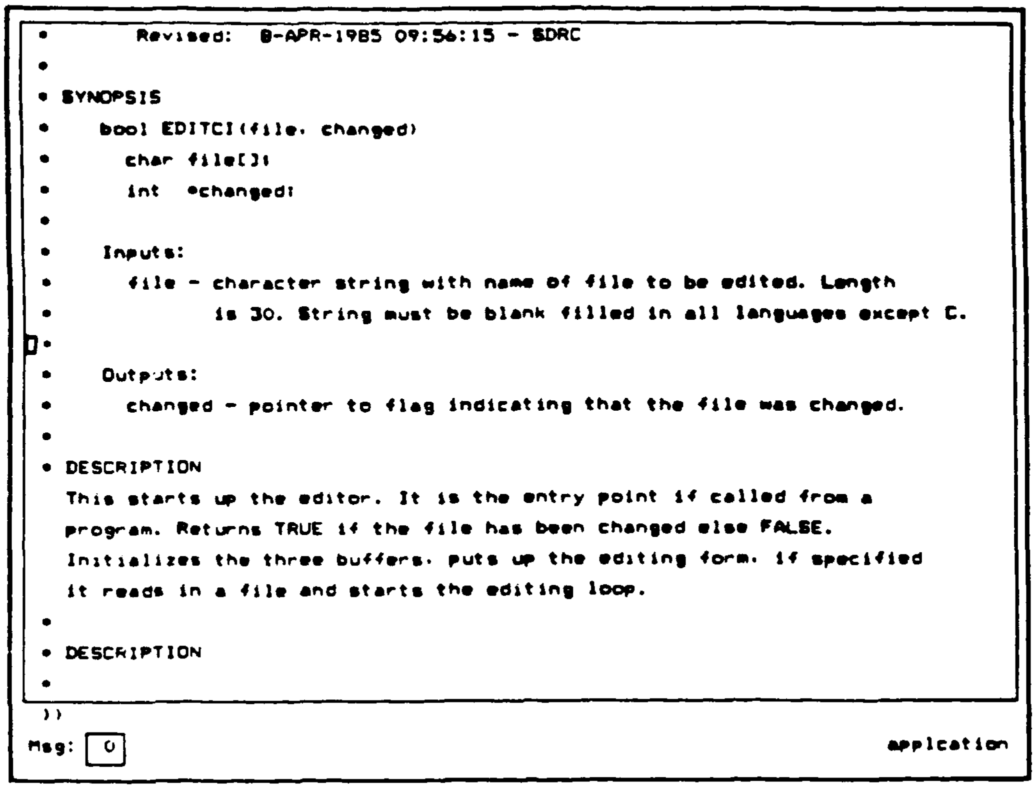
```
/* NAME
 *    EDITCI - EDIT Callable Interface
 *       Written: 23-OCT-1984 12:58:20 - SCGLANDORF
 *       Revised:  8-APR-1985 09:56:15 - SDRC
 *
 * SYNOPSIS
 *    bool EDITCI(file, changed)
 *      char file[];
 *      int  *changed;
 *
 *
 *    Outputs:
 *      changed - pointer to flag indicating that the file was changed.
 *
 * DESCRIPTION
   This starts up the editor. It is the entry point if called from a
   program. Returns TRUE if the file has been changed else FALSE.
   Initializes the three buffers, puts up the editing form, if specified
   it reads in a file and starts the editing loop.
 *
 * DESCRIPTION
 *
 ))
Msg: 0                                                      applcation
```

Figure A-23 Before Paste of Cut Buffer (3.2.1.1.10.3.a)



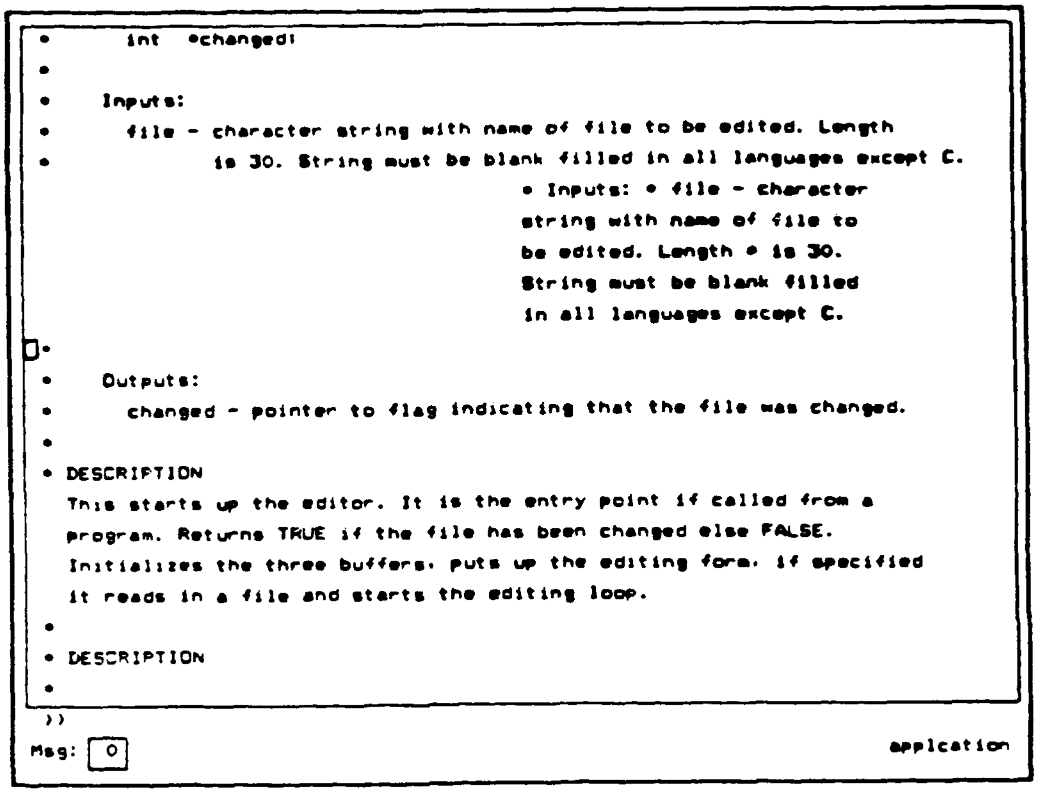
```
*       Revised:  8-APR-1985 09:56:15 - SDRC
*
* SYNOPSIS
*    bool EDITCI(file, changed)
*      char file[];
*      int  *changed;
*
*    Inputs:
*      file - character string with name of file to be edited. Length
*             is 30. String must be blank filled in all languages except C.
*
*    Outputs:
*      changed - pointer to flag indicating that the file was changed.
*
* DESCRIPTION
  This starts up the editor. It is the entry point if called from a
  program. Returns TRUE if the file has been changed else FALSE.
  Initializes the three buffers, puts up the editing form, if specified
  it reads in a file and starts the editing loop.
*
* DESCRIPTION
*
 ))
Msg: 0                                                        applcation
```

Figure A-24 After Paste of Cut Buffer (3.2.1.1.10.3.b)



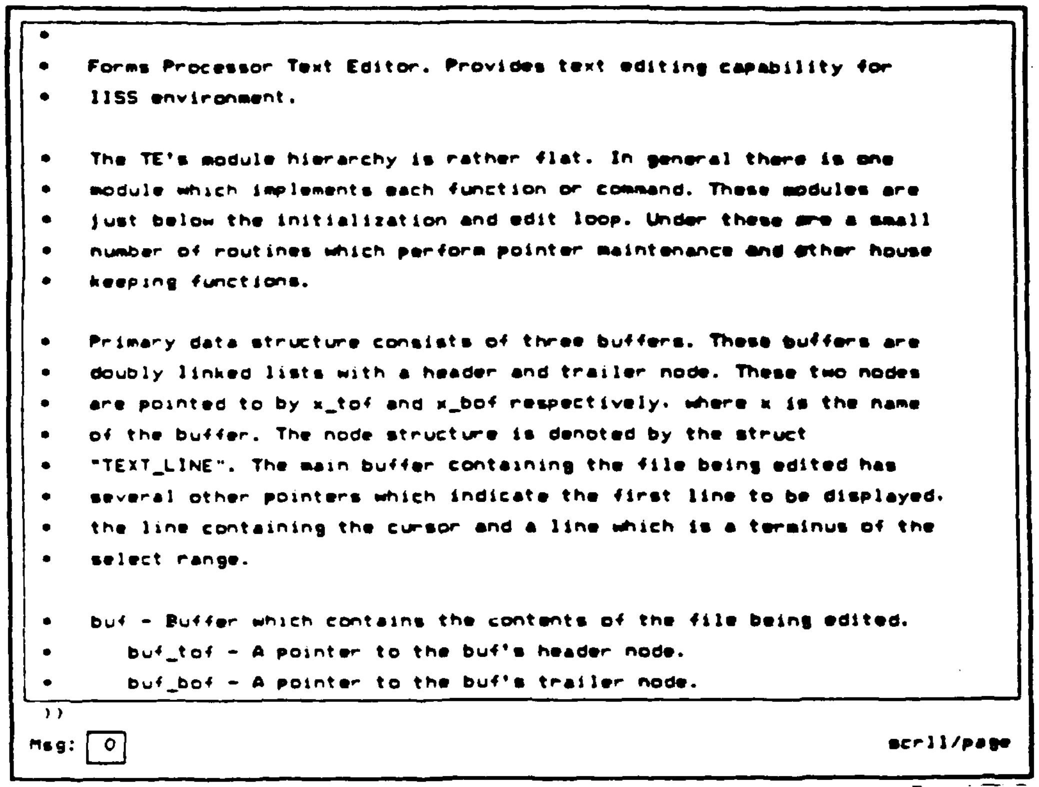
```
*       Revised:  8-APR-1985 09:56:15 - SDRC
*
* SYNOPSIS
*    bool EDITCI(file, changed)
*      char file[];
*      int  *changed;
*
*    Inputs:
*      file - character string with name of file to be edited. Length
*             is 30. String must be blank filled in all languages except C.
*
*    Outputs:
*      changed - pointer to flag indicating that the file was changed.
*
* DESCRIPTION
  This starts up the editor. It is the entry point if called from a
  program. Returns TRUE if the file has been changed else FALSE.
  Initializes the three buffers, puts up the editing form, if specified
  it reads in a file and starts the editing loop.
*
* DESCRIPTION
*
))
Msg: 0                                                       applcation
```

Figure A-25 Before Fill of Cut Buffer - margins changed to 40 70 (3.2.1.1.10.4.a)



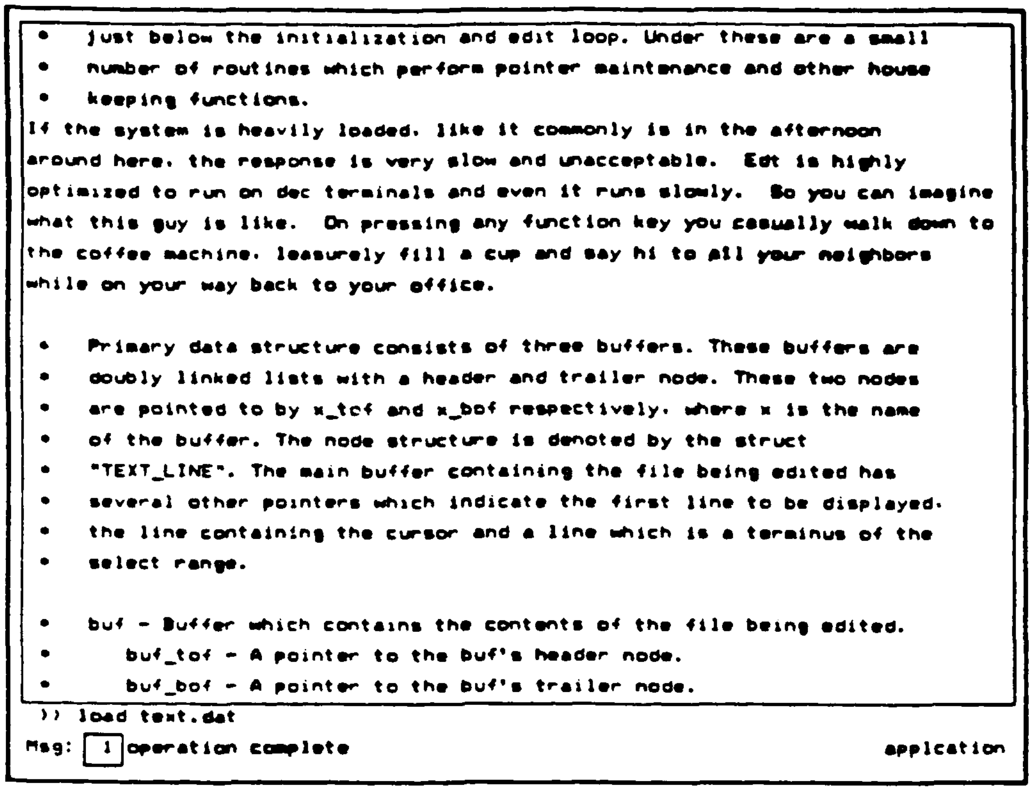
```
*       int  *changed;
*
*    Inputs:
*       file - character string with name of file to be edited. Length
*              is 30. String must be blank filled in all languages except C.
                                          * Inputs: * file - character
                                          string with name of file to
                                          be edited. Length * is 30.
                                          String must be blank filled
                                          in all languages except C.
□*
*    Outputs:
*       changed - pointer to flag indicating that the file was changed.
*
* DESCRIPTION
  This starts up the editor. It is the entry point if called from a
  program. Returns TRUE if the file has been changed else FALSE.
  Initializes the three buffers, puts up the editing form, if specified
  it reads in a file and starts the editing loop.
*
* DESCRIPTION
*
))
Msg: [0]                                                        applcation
```

Figure A-26 After Fill of Cut Buffer (3.2.1.1.10.4.b)



```
*
*   Forms Processor Text Editor. Provides text editing capability for
*   IISS environment.

*   The TE's module hierarchy is rather flat. In general there is one
*   module which implements each function or command. These modules are
*   just below the initialization and edit loop. Under these are a small
*   number of routines which perform pointer maintenance and other house
*   keeping functions.

*   Primary data structure consists of three buffers. These buffers are
*   doubly linked lists with a header and trailer node. These two nodes
*   are pointed to by x_tof and x_bof respectively, where x is the name
*   of the buffer. The node structure is denoted by the struct
*   "TEXT_LINE". The main buffer containing the file being edited has
*   several other pointers which indicate the first line to be displayed,
*   the line containing the cursor and a line which is a terminus of the
*   select range.

*   buf - Buffer which contains the contents of the file being edited.
*      buf_tof - A pointer to the buf's header node.
*      buf_bof - A pointer to the buf's trailer node.
))
Msg: 0                                                         scrll/page
```

Figure A-27 Before Load File (3.2.1.2.3.a)



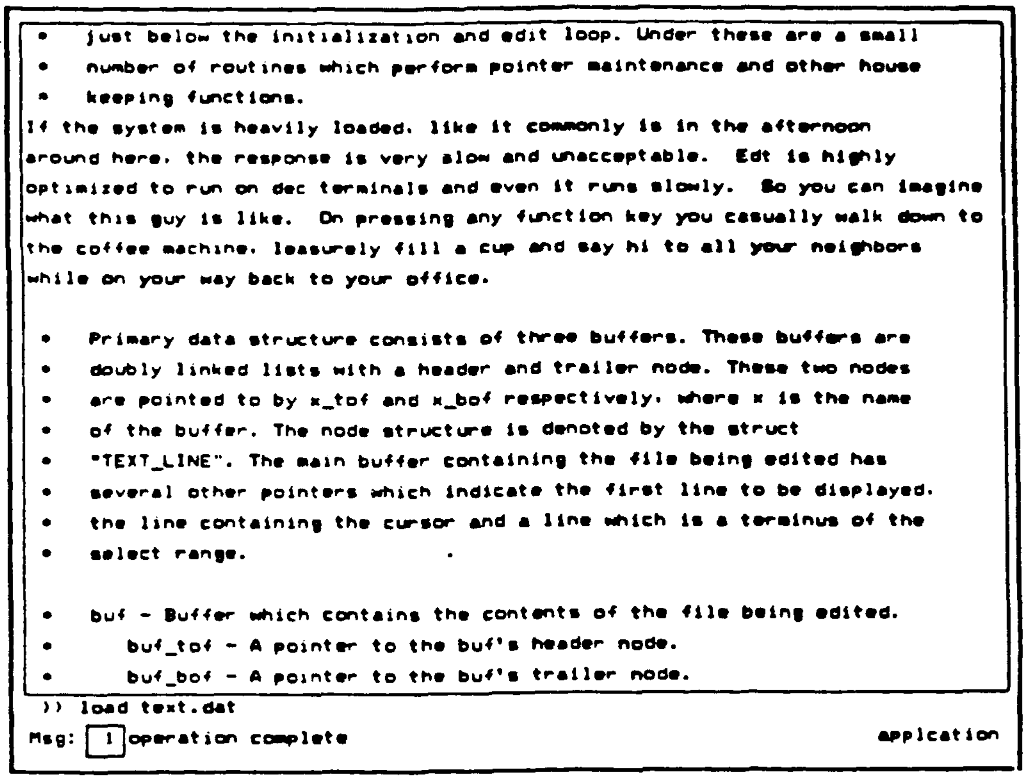
```
•   just below the initialization and edit loop. Under these are a small
•   number of routines which perform pointer maintenance and other house
•   keeping functions.
If the system is heavily loaded, like it commonly is in the afternoon
around here, the response is very slow and unacceptable.  Edt is highly
optimized to run on dec terminals and even it runs slowly.  So you can imagine
what this guy is like.  On pressing any function key you casually walk down to
the coffee machine, leasurely fill a cup and say hi to all your neighbors
while on your way back to your office.

•   Primary data structure consists of three buffers. These buffers are
•   doubly linked lists with a header and trailer node. These two nodes
•   are pointed to by x_tof and x_bof respectively, where x is the name
•   of the buffer. The node structure is denoted by the struct
•   "TEXT_LINE". The main buffer containing the file being edited has
•   several other pointers which indicate the first line to be displayed,
•   the line containing the cursor and a line which is a terminus of the
•   select range.

•   buf - Buffer which contains the contents of the file being edited.
•      buf_tof - A pointer to the buf's header node.
•      buf_bof - A pointer to the buf's trailer node.
 >> load text.dat
Msg: [1]operation complete                                        applcation
```

Figure A-28 After Load File (3.2.1.2.3.b)

APPENDIX B

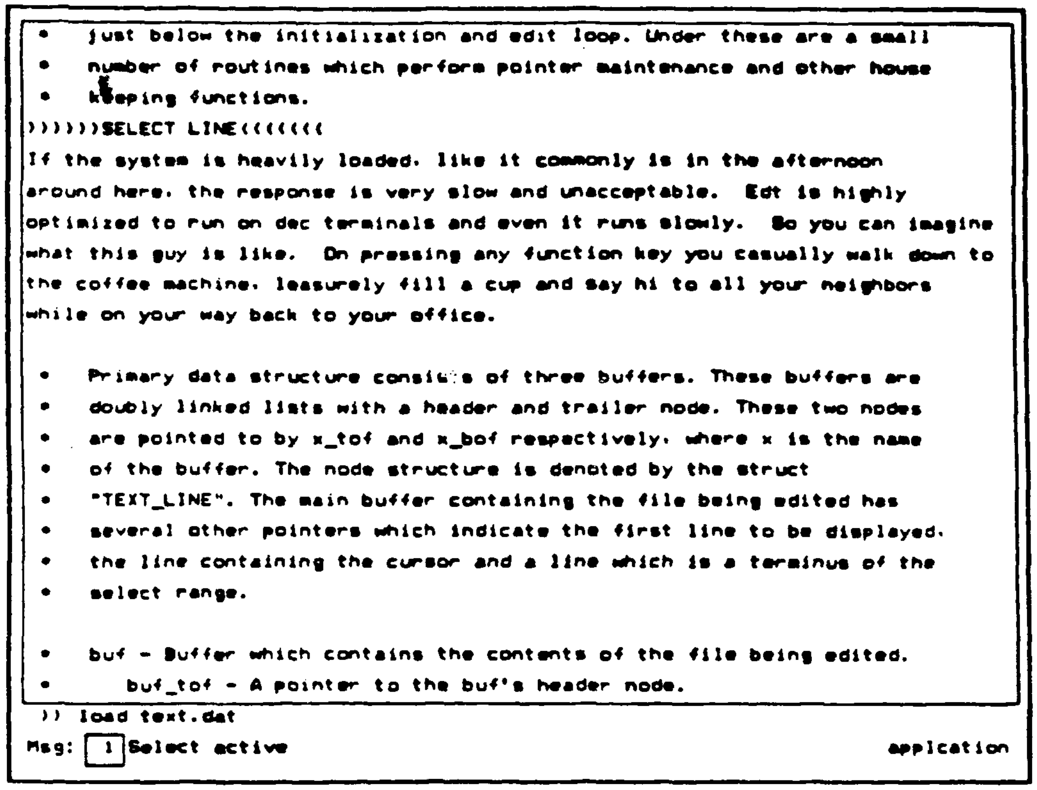
CUT AND PASTE SCENARIO

The following figures illustrate the operation of selecting a region of text to cut and then paste without and with fill. The figures are in reference to section 3.2.1.1.10.



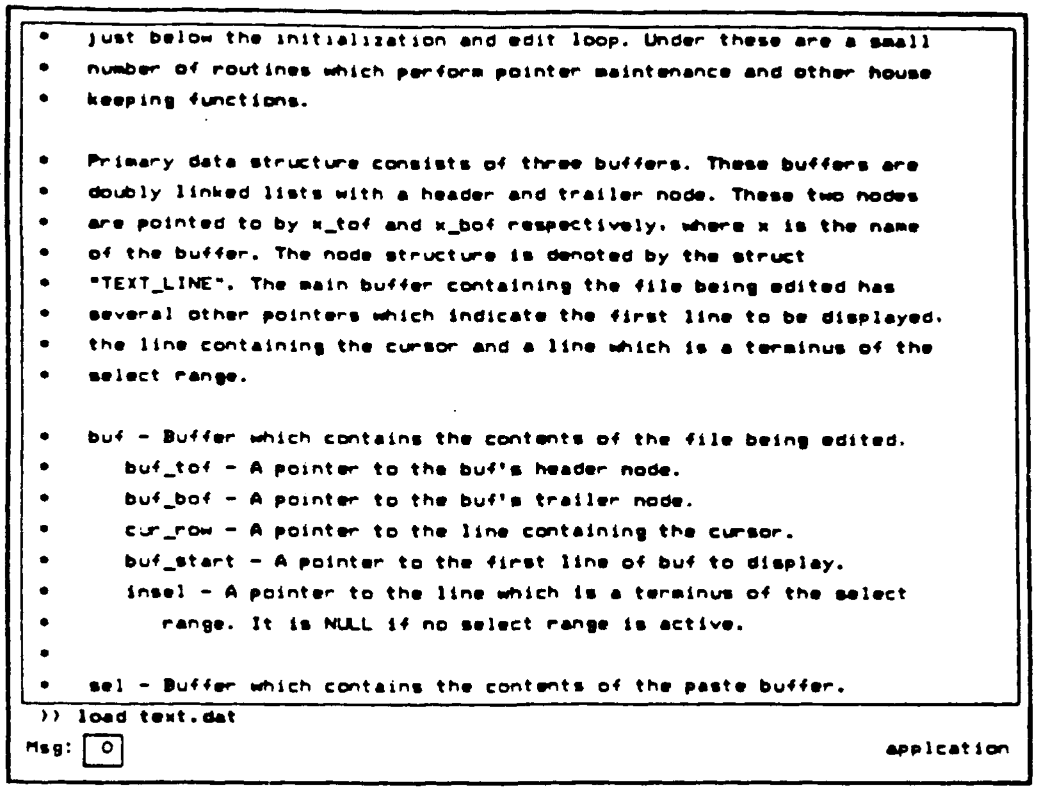
```
*   just below the initialization and edit loop. Under these are a small
*   number of routines which perform pointer maintenance and other house
*   keeping functions.
If the system is heavily loaded, like it commonly is in the afternoon
around here, the response is very slow and unacceptable.  Edt is highly
optimized to run on dec terminals and even it runs slowly.  So you can imagine
what this guy is like.  On pressing any function key you casually walk down to
the coffee machine, leasurely fill a cup and say hi to all your neighbors
while on your way back to your office.

*   Primary data structure consists of three buffers. These buffers are
*   doubly linked lists with a header and trailer node. These two nodes
*   are pointed to by x_tof and x_bof respectively, where x is the name
*   of the buffer. The node structure is denoted by the struct
*   "TEXT_LINE". The main buffer containing the file being edited has
*   several other pointers which indicate the first line to be displayed,
*   the line containing the cursor and a line which is a terminus of the
*   select range.

*   buf - Buffer which contains the contents of the file being edited.
*      buf_tof - A pointer to the buf's header node.
*      buf_bof - A pointer to the buf's trailer node.
 >> load text.dat
Msg: [1]operation complete                                      applcation
```

B-1 Before Select for Cut and Paste



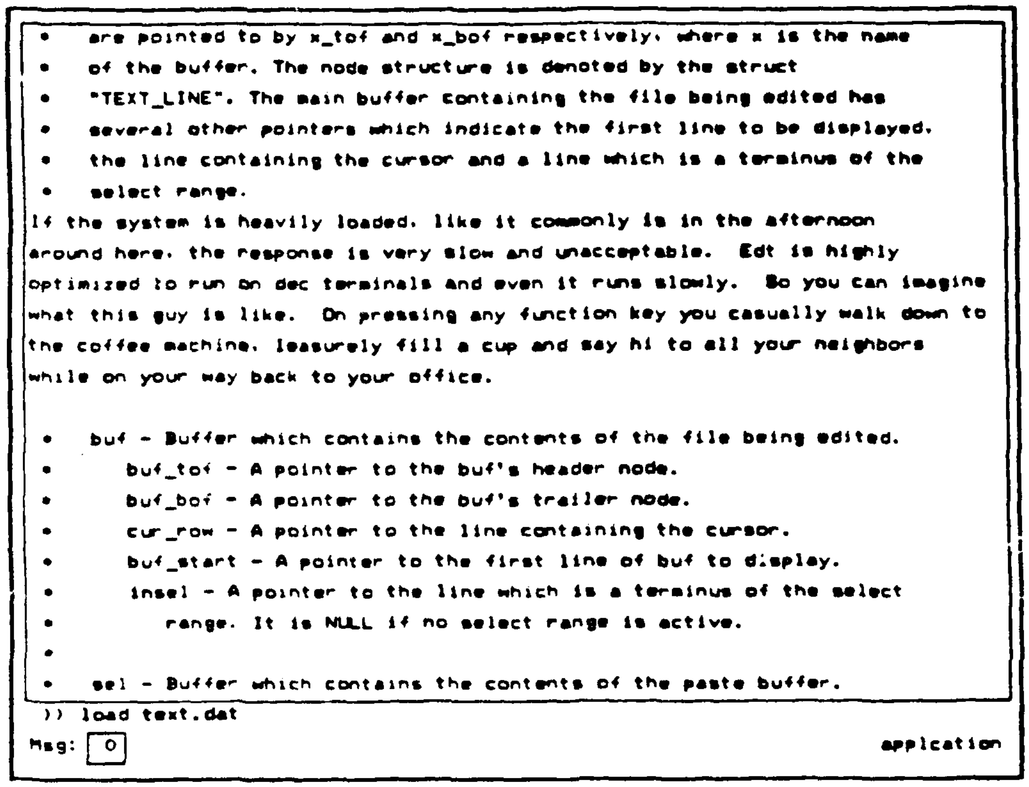
```
•    just below the initialization and edit loop. Under these are a small
•    number of routines which perform pointer maintenance and other house
•    keeping functions.
))))))SELECT LINE(((((((
If the system is heavily loaded, like it commonly is in the afternoon
around here, the response is very slow and unacceptable.  Edt is highly
optimized to run on dec terminals and even it runs slowly.  So you can imagine
what this guy is like.  On pressing any function key you casually walk down to
the coffee machine, leasurely fill a cup and say hi to all your neighbors
while on your way back to your office.

•    Primary data structure consists of three buffers. These buffers are
•    doubly linked lists with a header and trailer node. These two nodes
•    are pointed to by x_tof and x_bof respectively, where x is the name
•    of the buffer. The node structure is denoted by the struct
•    "TEXT_LINE". The main buffer containing the file being edited has
•    several other pointers which indicate the first line to be displayed,
•    the line containing the cursor and a line which is a terminus of the
•    select range.

•    buf - Buffer which contains the contents of the file being edited.
•       buf_tof - A pointer to the buf's header node.
 )) load text.dat
Msg: 1 Select active                                           application
```

B-2 After Select, Before Delete Line



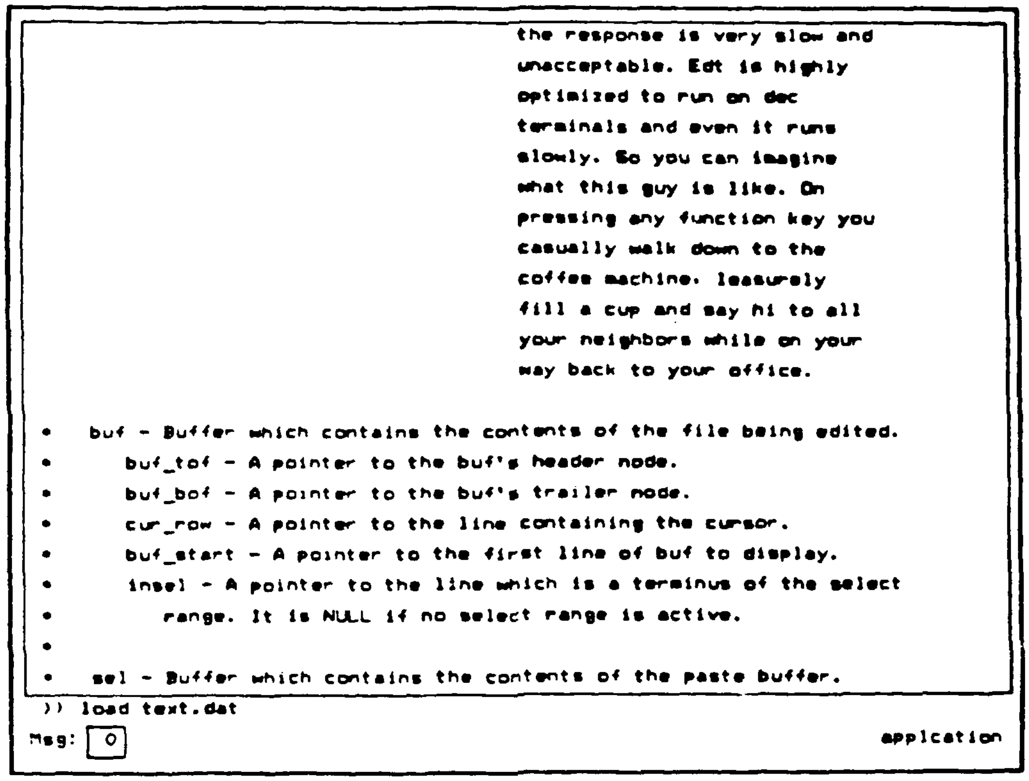
```
•  just below the initialization and edit loop. Under these are a small
•  number of routines which perform pointer maintenance and other house
•  keeping functions.

•  Primary data structure consists of three buffers. These buffers are
•  doubly linked lists with a header and trailer node. These two nodes
•  are pointed to by x_tof and x_bof respectively, where x is the name
•  of the buffer. The node structure is denoted by the struct
•  "TEXT_LINE". The main buffer containing the file being edited has
•  several other pointers which indicate the first line to be displayed,
•  the line containing the cursor and a line which is a terminus of the
•  select range.

•  buf - Buffer which contains the contents of the file being edited.
•     buf_tof - A pointer to the buf's header node.
•     buf_bof - A pointer to the buf's trailer node.
•     cur_row - A pointer to the line containing the cursor.
•     buf_start - A pointer to the first line of buf to display.
•     insel - A pointer to the line which is a terminus of the select
•        range. It is NULL if no select range is active.
•
•  sel - Buffer which contains the contents of the paste buffer.
)) load text.dat
Msg: [0]                                                  application
```

B-3 After Delete Line, Before Paste



```
•    are pointed to by x_tof and x_bof respectively, where x is the name
•    of the buffer. The node structure is denoted by the struct
•    "TEXT_LINE". The main buffer containing the file being edited has
•    several other pointers which indicate the first line to be displayed,
•    the line containing the cursor and a line which is a terminus of the
•    select range.
If the system is heavily loaded, like it commonly is in the afternoon
around here, the response is very slow and unacceptable.  Edt is highly
optimized to run on dec terminals and even it runs slowly.  So you can imagine
what this guy is like.  On pressing any function key you casually walk down to
the coffee machine, leasurely fill a cup and say hi to all your neighbors
while on your way back to your office.

•    buf - Buffer which contains the contents of the file being edited.
•       buf_tof - A pointer to the buf's header node.
•       buf_bof - A pointer to the buf's trailer node.
•       cur_row - A pointer to the line containing the cursor.
•       buf_start - A pointer to the first line of buf to display.
•       insel - A pointer to the line which is a terminus of the select
•          range. It is NULL if no select range is active.
•
•    sel - Buffer which contains the contents of the paste buffer.
 )) load text.dat
Msg: [0]                                                          applcation
```

B-4 After Paste, Before Fill



```
                                              the response is very slow and
                                              unacceptable. Edt is highly
                                              optimized to run on dec
                                              terminals and even it runs
                                              slowly. So you can imagine
                                              what this guy is like. On
                                              pressing any function key you
                                              casually walk down to the
                                              coffee machine. leasurely
                                              fill a cup and say hi to all
                                              your neighbors while on your
                                              way back to your office.

•   buf - Buffer which contains the contents of the file being edited.
•     buf_tof - A pointer to the buf's header node.
•     buf_bof - A pointer to the buf's trailer node.
•     cur_row - A pointer to the line containing the cursor.
•     buf_start - A pointer to the first line of buf to display.
•     insel - A pointer to the line which is a terminus of the select
•       range. It is NULL if no select range is active.
•
•   sel - Buffer which contains the contents of the paste buffer.
 )) load text.dat
Msg: 0                                                           applcation
```

B-5 After Fill

APPENDIX C

Keypad layouts for VT100 in application mode

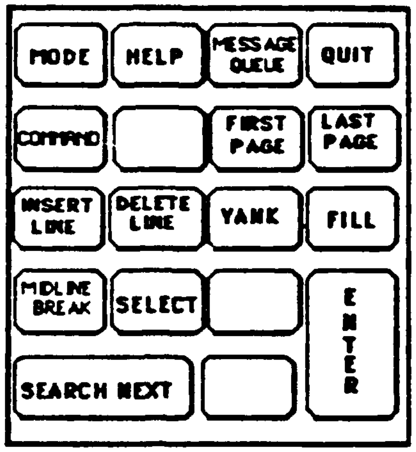


Figure C-1 Application Mode

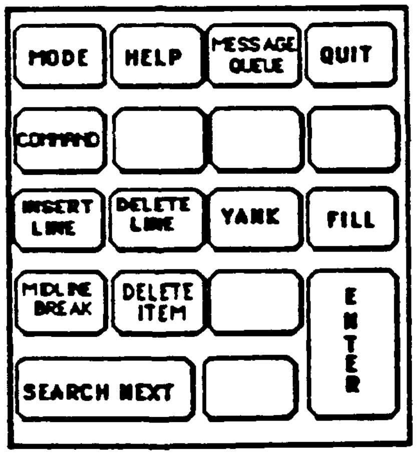


Figure C-2 Text Edit Mode